User Manual

3-Heights™ PDF to PDF/A Converter API

Version 4.5

PDF/A compliant

PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction . . . 1
1.1 Description . . . 1
1.2 Functions . . . 2
1.3 Interfaces . . . 3
1.4 Operating Systems . . . 3
1.5 How to Best Read this Manual . . . 3
1.6 Digital Signatures . . . 3
What is an Electronic Signature? . . . 5
How to Create Electronic Signatures . . . 6
2 Installation and Deployment . . . 7
2.1 Download and Installation . . . 7
2.2 Unix . . . 9
2.3 Interfaces . . . 10
2.4 Uninstall, Install a New Version . . . 12
3 License Management . . . 13
3.1 Graphical License Manager Tool . . . 13
3.2 Command Line License Manager Tool . . . 14
3.3 License Key Storage . . . 14
4 Programming Interfaces . . . 15
4.1 Visual Basic 6 . . . 15
4.2 .NET . . . 16
4.3 Java Interface . . . 18
4.4 ASP . . . 18
5 User's Guide . . . 19
5.1 Process Description . . . 19
5.2 What is PDF/A? . . . 21
5.3 Color Spaces . . . 22
5.4 Fonts . . . 24
5.5 Cryptographic Provider . . . 25
PKCS#11 Provider . . . 26
Microsoft CryptoAPI Provider . . . 27
3-Heights™ Signature Creation and Validation Service . . . 30
SwissSign Personal Signing Service . . . 32
SwissSign SuisseID Signing Service . . . 33
QuoVadis sealsign . . . 34
Swisscom All-in Signing Service . . . 35
5.6 How to Create Digital Signatures . . . 36
How to Create a PAdES LTV Signature . . . 36
How to Create a Visual Appearance of a Signature . . . 37
Guidelines for Mass Signing . . . 37
Miscellaneous . . . 38
5.7 How to Validate Digital Signatures . . . 40
6 Reference Manual . . . 42
6.1 Pdf2PdfAPI Interface . . . 43
AddAssociatedFile . . . 43
AddEmbeddedFile . . . 43
AddZUGFeRDXml . . . 43
AddFontDirectory . . . 43
AnalyzeOnly . . . 44
BeginSession . . . 44
CertificateName . . . 44
ColorSpaceProfile . . . 45
Compliance . . . 45
ConversionErrors . . . 45
ConversionErrorMask . . . 46
Convert . . . 46

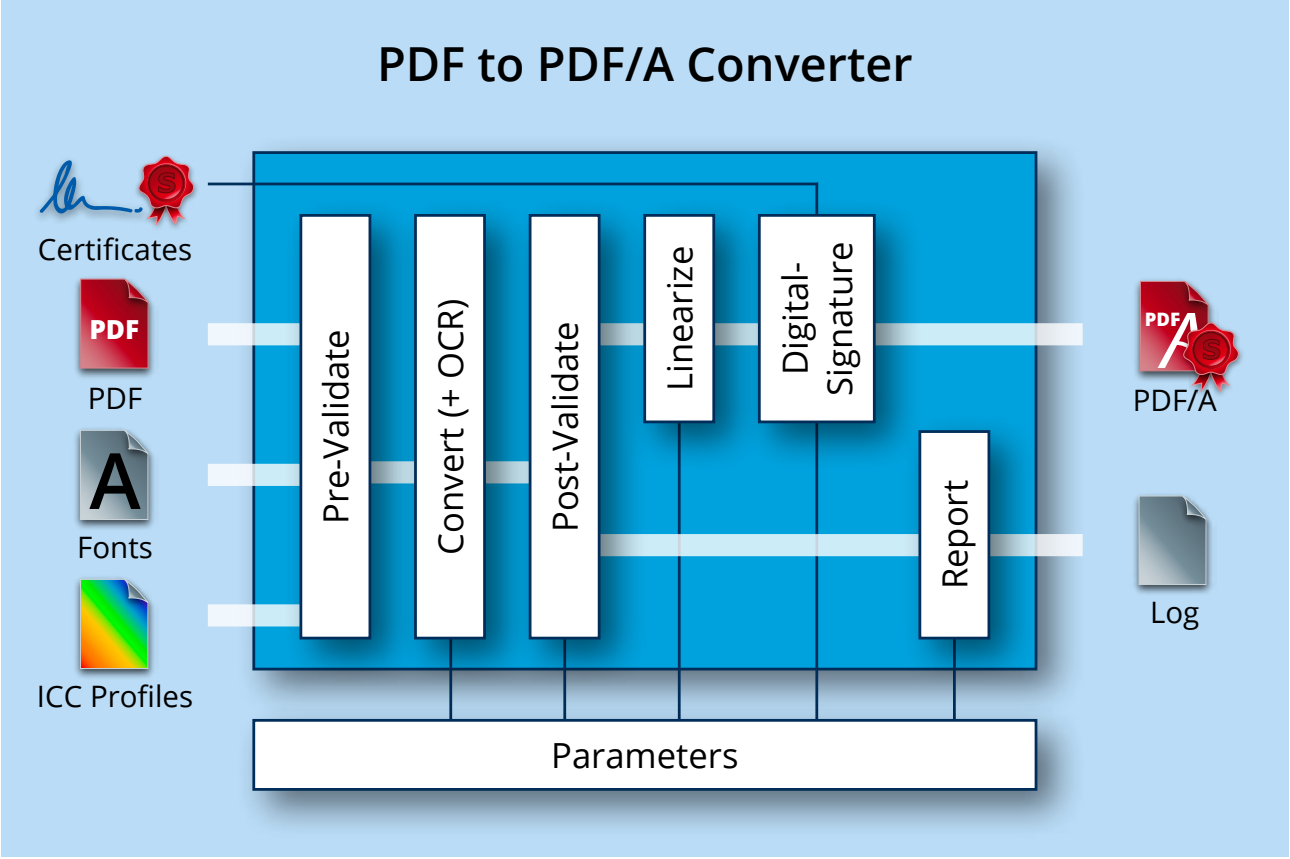
ConvertAlways . . . 47
ConvertMem . . . 47
EmbedT1asCFF . . . 47
EmbedRevocationInfo . . . 48
EndSession . . . 48
ErrorCode . . . 48
ExportText . . . 48
FlattenSignatures . . . 49
GetOCRPluginCount . . . 49
GetOCRPluginName . . . 50
ImageQuality . . . 50
InfoEntry . . . 50
Issuer . . . 51
Linearize . . . 51
NoCache . . . 51
OCRBitonalRecognition . . . 51
OCRMode . . . 52
OCRReembedImages . . . 52
OCRDeskewImage . . . 52
OCREmbedBarcodes . . . 52
OCRRotatePage . . . 53
OCRResolutionDPI . . . 53
OCRThresholdDPI . . . 53
OutputIntentProfile . . . 53
PostAnalyze . . . 53
Provider . . . 54
ProxyURL . . . 54
ProxyCredentials . . . 54
Remove Signature . . . 54
ReportDetails . . . 55
ReportSummary . . . 55
SetMetadata . . . 56
SetOCREngine . . . 56
SetOCRLanguages . . . 56
SetOCRParams . . . 57
SetSessionProperty . . . 57
SetSignatureRectangle . . . 57
SetToUnicodeFile . . . 57
SignatureImageFileName . . . 58
SignatureLocation . . . 58
SignaturePageNo . . . 58
SignatureReason . . . 58
SignatureSerialNumber . . . 58
SignerContactInfo . . . 58
SignerFingerprint . . . 59
SignerFingerprintStr . . . 59
Store . . . 59
StoreLocation . . . 59
SubsetFonts . . . 59
TimeStampCredentials . . . 60
TimeStampURL . . . 60
Terminate . . . 60
TestSession . . . 60
6.2 Enumerations . . . 61
TPDFCompliance . . . 61
TPDFErrorCode . . . 61
TPDFConversionError . . . 62
7 Log File . . . 64
8 Licensing and Copyright . . . 65
9 Contact . . . 65

1 Introduction

1.1 Description

The 3-Heights™ PDF to PDF/A Converter API converts PDF files into PDF/A files. PDF/A has been acknowledged world-wide as the ISO standard for long-term archiving since 2005. The tool analyzes and converts the input file, applying a digital signature where required.

The integrated validator then optionally checks conformity once again. This product is robust and powerful and therefore predestined for archive migrations of any size.



1.2 Functions

The 3-Heights™ PDF to PDF/A Converter API accepts files from many different applications and automatically converts them into PDF/A. The level of conformity can be set to Level A or Level B. ICC color profiles for device-dependent color profiles and font types are embedded in the document. There is an option to provide the entire character set for fonts (no subsetting) to facilitate editing at a later stage. Missing fonts are reproduced as close to the original as possible via font recognition. Metadata can be generated automatically or added from external sources. The tool also detects and automatically repairs problems typical of the PDF format. A digital signature can be applied and a conformity check carried out at the end of the process. The optional OCR Add-On and linearization for fast web display are valuable additional functions.

Features

- Conversion (PDF/A-1, PDF/A-2, PDF/A-3)
- Selectable level of conformity
- Embedding ICC color profiles for device-dependent color spaces
- Replace and subset fonts
- Validation
- File analysis and repair
- Conversion reporting
- Digital signatures, PDF/A-compliant
- Configure the virtual appearance of the signature (page, size, color, position, text, background image, etc.)
- Write the application log to a log file
- Enforce conversion even if the file is unconvertible
- Read input and output document from file or memory
- Metadata management
- Read encrypted input files
- Encryption with access authorizations (not for PDF/A)
- Linearization (fast web display)
- JBIG2 compression
- JPEG2000 compression
- Conversion of embedded and attached files (PDF/A-2 and later)
- Colorants management (PDF/A-2 and later)
- OCR (optional)
- List OCR plug-Ins
- Set the OCR language
- Add embedded files (PDF/A-2) and associated files (PDF/A-3)
- Embedded XML invoice data conforming to the ZUGFeRD specification (PDF/A-3).

Formats

Input Formats

- PDF 1.x (PDF 1.4, PDF 1.5, etc)

Target Formats:

- PDF/A-1a, PDF/A-1b
- PDF/A-2a, PDF/A-2b, PDF/A-2u
- PDF/A-3a, PDF/A-3b, PDF/A-3u

Compliance

- Standards: ISO 19005-1 (PDF/A-1), ISO 19005-2 (PDF/A-2), ISO 19005-3 (PDF/A-3), ISO 32000 (PDF 1.7), PAdES Part 2
- Quality assurance: Isartor test suite

1.3 Interfaces

The following interfaces are available:

- C
- Java
- .NET
- COM

1.4 Operating Systems

- Windows XP, Vista, 7, 8, 8.1 - 32 and 64 bit
- Windows Server 2003, 2008, 2008 R2, 2012, 2012 R2 - 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 32 bit or HP-UX 11i and later ia64 (Itanium) 64 bit
- IBM AIX 5.1 and later (64 bit)
- Linux (32 and 64 bit)
- Mac OS X 10.4 and later (32 and 64 bit)
- Sun Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later 32 bit or FreeBSD 9.3 and later 64 bit (on request)

1.5 How to Best Read this Manual

If you are reading this manual for the first time, i.e. would like to evaluate the software, the following steps are suggested.

1. Read the chapter Introduction to verify this product meets your requirements.
2. Identify what interface your programming language uses.
3. Read and follow the instructions in the chapter Installation And Deployment.
4. In the chapter Programming Interfaces find your programming language. Please note that not every language is covered in this manual.
   For many programming languages there is sample code available. For a start it is generally best to refer to these samples rather than writing code from scratch.
5. (Optional) Read the chapter User's Guide for general information about the API. Read Programmer's Reference for specific information about the functions of the API.

1.6 Digital Signatures

Overview

Digital signature is a large and slightly complex topic. This manual gives an introduction to digital signatures and describes how the 3-Heights™ PDF to PDF/A Converter API is used to apply them. It does however not describe all the technical details.

Terminology

Digital Signature is a cryptographic technique of calculating a number (a digital signature) for a message. Creating a digital signature requires a private key from a certificate. Validating a digital signature and its authorship requires a public key. Digital Signature is a technical term.

Electronic Signature is a set of electronic data that is merged or linked to other electronic data in order to authenticate it. Electronic Signatures can be created by means of a digital signature or other techniques. Electronic Signature is a legal term.

Table: Abbreviations	
CA	Certification Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
HSM	Hardware Security Module
OCSP	Online Certificate Status Protocol
PKCS	Public Key Cryptography Standards
QES	Qualified Electronic Signature
TSA	Time-stamp Authority
TSP	Time-stamp Protocol

Why Digitally Signing?

The idea of applying a digital signature in PDF is very similar to a handwritten signature: A person reads a document and signs it with their name. In addition to the name, the signature can contain further optional information, such as the date and location. A valid electronic signature is a section of data that can be used to:

- Ensure the integrity of the document
- Authenticate the signer of the document
- Prove existence of file prior to date (time-stamp)

Digitally signing a document requires a certificate and its private key. How to access and use a certificate is described in the chapter Cryptographic Provider.

In a PDF document, a digital signature consists of two parts:

- *A PDF related part*: This part consists of the PDF objects required to embed the signature into the PDF document. This part depends on the signature type (Document Signature, MDP Signature, see table below). Information such as name of the signer, reason, date, location is stored here. The signature may optionally have a visual appearance on a page of the PDF document, which can contain text, graphics and images. This part of the signature is entirely created by the 3-Heights™ PDF to PDF/A Converter API.
- *A cryptographic part*: A digital signature is based on a cryptographic checksum (hash value) calculated from the content of the document that is being signed. If the document is modified at a later time, the computed hash value is no longer correct and the signature becomes invalid, i.e. the validation will fail and will report that the document has been modified since the signature was applied. Only the owner of the certificate and its private key is able to sign the document. However, anybody can verify the signature with the public key contained in the certificate. This part of the signature requires a cryptographic provider for some cryptographic data and algorithms.

The 3-Heights™ PDF to PDF/A Converter API supports the following types of digital signatures:

- *Document Signature*: Check the integrity of the signed part of the document and authenticate the signer's identity. One or more signatures can be applied. A signed document can be modified and saved by incremental update. The state of the document can be re-created as it existed at the time of signing.
- *MDP (Modification detection and prevention) Signature*: Enable detection of disallowed changes specified by the author. A document can contain only one MDP signature; it must be the first in the document. Other document signatures may be present.
- *Document Time-stamp Signature*: Establish the exact content of the file at the time indicated by the Time-stamp. One or more document Time-stamp signatures can be applied. A signed document can be modified and saved by incremental update.

1.6.1 What is an Electronic Signature?

There are different types of electronic signatures, which normally are defined by national laws, and therefore are different for different countries. The type of electronic signatures required in a certain process is usually defined by national laws. Quite advanced in this manner are German-speaking countries where such laws and an established terminology exist. The English terminology is basically a translation from German. Three types of electronic signatures are distinguished:

- Simple Electronic Signature “Einfache Elektronische Signatur”
- Advanced Electronic Signature “Fortgeschrittene Elektronische Signatur”
- Qualified Electronic Signature (QES) “Qualifizierte Elektronische Signatur”

All applied digital signatures are PDF/A and PAdES compliant.

Simple Electronic Signature

A simple electronic signature requires any certificate that can be used for digital signing. The easiest way to retrieve a certificate, which meets that requirement, is to create a so called self-signed certificate. Self-signed means it is signed by its owner, therefore the issuer of the certificate and the approver of the legitimacy of a document signed by this certificate is the same person.

Example: Anyone could create a self-signed certificate issued by “Peter Pan” and issued to “Peter Pan”. Using this certificate one is able to sign in the name of “Peter Pan”.

If a PDF document is signed with a simple electronic signature and the document is changed after the signature has been applied, the signature becomes invalid. However, the person who applied the changes, could at the same time (maliciously) also remove the existing simple electronic signature and - after the changes - apply a new, equally looking Simple Electronic Signature and falsify its date. As we can see, a simple electronic signature is neither strong enough to ensure the integrity of the document nor to authenticate the signer.

This drawback can be overcome using an Advanced or Qualified Electronic Signature.

Advanced Electronic Signature

Requirements for advanced certificates and signatures vary depending on the country where they are issued and used.

An advanced electronic signature is based on an advanced certificate that is issued by a recognized certificate authority (CA) in this country, such VeriSign, SwissSign, QuoVadis. In order to receive an advanced certificate, its owner must prove his identity, e.g. by physically visiting the CA and presenting his passport. The owner can be an individual, a legal person or another entity.

An advanced certificate contains the name of the owner, the name of the CA, its period of validity and other information.

The private key of the certificate is protected by a PIN, which is only known to its owner.

This brings the following advantages over a simple electronic signature:

- The signature authenticates the signer.
- The signature ensures the integrity of the signed content.

Qualified Electronic Signature

Requirements for qualified certificates and signatures vary depending on the country where they are issued and used.

A Qualified Electronic Signature is similar to an advanced electronic signature, but has higher requirements. The main differences are:

- It is based on a qualified certificate, which is provided as a hardware token (USB stick, smart card).
- For every signature it is required to enter the PIN code manually. This means that only one signature can be applied at a time.
- Certificate revocation information (OCSP/CRL) can be acquired from an online service. The response (valid, revoked, etc.) must be embedded in the signature.
- A time-stamp (TSP) that is acquired from a trusted time server (TSA) may be required.

This brings the following advantages over an advanced electronic signature:

- The signature ensures the certificate was valid at the time when the document was signed (due to the embedding of the OCSP/CRL response).
- The signature ensures the integrity of the time of signing (due to the embedding of the time-stamp).
- Legal processes that require a QES are supported.

Note that a Time-stamp can be added to any type of signature. OCSP/CRL responses are also available for some advanced certificates.

1.6.2 How to Create Electronic Signatures

Preparation Steps

1. Identify whether an advanced or a qualified signature is required. For most automated processes an advanced signature is sufficient.
2. Acquire a corresponding certificate from a CA. Note that some CA offer USB sticks or smart cards that contain both, an advanced and a qualified certificate.
3. Setup and configure the certificate's Cryptographic Provider
   - In case the certificate resides on hardware such as an USB token or a Smart Card, the required middleware (driver) needs to be installed.
   - In case the certificate is a soft certificate, it must be imported into the certificate store of a cryptographic provider.
4. Optional: Acquire access to a trusted time server (TSA) (e.g. from the CA of your signing certificate).
5. Apply the signature by providing the following information:
   - Values for the selection of the signing certificate (e.g. the name of the certificate)
   - The Cryptographic Provider where the certificate is located
   - Optional: Time-stamp service URL (e.g. "http://server.mydomain.com:80/tsa")
   - Optional: Time-stamp service credentials (e.g. username:password)
   - Optional: Embed revocation information (default: true)
   - Optional: Visual appearance of the signature on a page of the document (e.g. an image).

Example: Steps to Add an Electronic Signature

The 3-Heights™ PDF to PDF/A Converter API applies PDF/A compliant signatures. This means if a PDF/A document is digitally signed, it remains PDF/A compliant.

In order to add an electronic signature with the 3-Heights™ PDF to PDF/A Converter API the following steps need to be done:

1. Create a new Pdf2PdfAPI object
2. As value of the Pdf2PdfAPI's name, the name of the certificate that is to be used must be provided. The name of the certificate corresponds to the value "Issued to:".
3. If the certificate's private key is PIN protected, the PIN can be passed in the provider configuration.
4. Additional parameters can now be set such as the reason why the signature is applied, etc.

In Visual Basic the four steps above look like this:

```
Dim conv As New Pdf2PdfAPI
conv.CertificateName = "Philip Renggli"
conv.Provider = "cvp11.dll;0;secret-pin"
conv.Reason = "I reviewed the document"                ' optional
conv.TimeStampURL = "http://server.mydomain.com:80/tsa" ' optional
conv.Convert "input.pdf", "password", "output.pdf", "log.txt"
```

Of course, you would use your own name instead. The name of the certificate is defined by its common name (CN), which is displayed as "issued to" in the Windows Certificate Store.

Note: On Non-Windows platforms additional settings are required to be set, such as fonts.

The visual appearance of the digital signature on a page of the resulting output-document looks as shown below:

Philip Renggli
Digitally signed by
Philip Renggli
Reason: I reviewed the document
Time: D:20061211132331

2 Installation and Deployment

2.1 Download and Installation

The installation of the software requires the following steps.

1. You need administrator rights to install this software.
2. Log in to your download account at http://www.pdf-tools.com. Select the product "PDF to PDF/A Converter API ". If you have multiple versions available, select an SDK version. The download account will show you one or multiple download links. If you have no active downloads available or cannot log in, please contact pdfsales@pdf-tools.com for assistance.
   You will find product version of different builds available. We always suggest using a so called "Final Release" version, which is a well tested and stable version and labeled with "final". Other versions are called "Pre-Release" and they normally contain new features and bug-fixes. We suggest using "Pre-Release" versions for evaluation and if you explicitly need a new feature or specific bug fix.

   There are 32 and 64 bit versions available. The 32 bit version runs on both, 32 and 64 bit platforms. There is a zipped MSI (*MSI.zip) and a ZIP (*.zip) version available. The MSI (Microsoft Installer) provides an installation routine that installs and uninstalls the product for you. The ZIP version allows you to select and install everything individually. Download the version you wish to install.
3. If you select an MSI version, extract the MSI, start it and follow the steps in the installation routine. No further steps are needed.
   If you are using the ZIP version, follow the steps below.
4. Open the ZIP archive. Check the appropriate option to preserve file paths (folder names) and unzip the archive to a local folder (e.g. *C:\program files\pdf-tools*).
5. The unzip process now creates the following subdirectories:

   bin: Contains the runtime executable binary code and the .NET assemblies

   bin\fonts: Contains the fonts "Symbol" and "ZapfDingbats" and the font mapping file

   bin\icc: Contains color profile and link to download more color profiles

   doc: Contains documentation files

   include: Contains header files to include in your C / C++ project

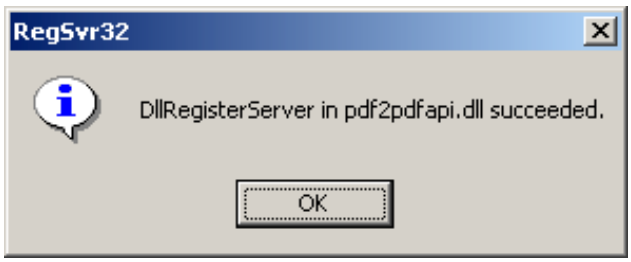
   jar: Contains the Java wrapper when using JNI

   lib: Contains the object file library to include in your C / C++ project

   samples: Contains samples in various programming languages
6. COM interface only: Before you can use the 3-Heights™ PDF to PDF/A Converter API component in your COM application program you have to register the component using the `regsvr32.exe` program that is provided with the Windows operating system; it resides in the directory System32. If you are using a newer operating

system, such as Vista or Windows 7, start the command prompt as Administrator. The following screenshot shows the registration of the PDF to PDF/A Converter API DLL.

If you are using a 64-bit operating system and would like to register the 32-bit version of the 3-Heights™ PDF to PDF/A Converter API, you need to use the regsvr32 from the directory `%SystemRoot%\SysWOW64` instead of `%SystemRoot%\System32`.[1]
If the registration process succeeds, the following dialog window is displayed:



The registration can also be done silently (e.g. for deployment) using the switch `/s`.
The other DLLs do not need to be registered, but for simplicity it is suggested that they are in the same directory as the *Pdf2PdfAPI.dll*.

7. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interfaces.
8. Make sure your platform meets the requirements regarding color spaces and fonts described in chapters Color Profiles and Fonts respectively.
9. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.

2.2 Unix

This section describes installation steps required on all Unix platforms, which includes Linux, Mac OS X, Sun Solaris, IBM AIX, HP-UX, FreeBSD and others.

The Unix version of the 3-Heights™ PDF to PDF/A Converter API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the shared libraries and other files that come with the 3-Heights™ PDF to PDF/A Converter API:

Table: File Description

Name	Description
`bin/libPdf2PdfAPI.so`	This is the shared library that contains the main functionality. The file extension varies depending on the UNIX platform.
`doc/*.*`	Documentation
`bin/CNVA.jar`	Java API archive.
`include/*.h`	Contains files to include in your C/C++ Project.

Example code written in different programming languages are available at product page of the PDF Tools AG website (http://www.pdf-tools.com).

[1] Otherwise you get the following message: *"LoadLibrary("Pdf2PdfAPI.dll") failed - The specified module could not be found."*

All Unix Platforms

1. Unpack the archive in an installation directory, e.g. */opt/pdf-tools.com/*
2. Copy or link the shared object into one of the standard library directories, e.g:
   `ln -s /opt/pdf-tools.com/bin/libPdf2PdfAPI.so /usr/lib`
3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*: `ldd libPdf2PdfAPI.so`
   - *On AIX*: `dump -H libPdf2PdfAPI.so`

   In case you have not installed the GNU shared libraries yet, proceed as follows:
   (a) Go to http://www.pdf-tools.com and navigate to "Support" → "Resouces".
   (b) Download the GNU shared libraries for your platform.
   (c) Extract the archive and copy or link the libraries into your library directory, e.g */usr/lib* or */usr/lib64*.
   (d) Verify that the GNU shared libraries required by the product are available on your system now.
4. Optionally register your license key using the Command Line License Manager Tool.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interfaces.
6. Make sure your platform meets the requirements regarding color spaces and fonts described in chapters Color Profiles and Fonts respectively.
7. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.

MAC OS/X

The shared library must have the extension .jnilib for use with Java. We suggest that you create a file link for this purpose by using the following command:

```
ln libPdf2PdfAPI.dylib libPdf2PdfAPI.jnilib
```

2.3 Interfaces

The 3-Heights™ PDF to PDF/A Converter API provides four different interfaces. The installation and deployment of the software depend on the interface you are using. The table below shows the supported interfaces and examples with which programming languages they can be used.

Table: Interfaces

Interface	Programming Languages
`.NET`	The MS software platform .NET can be used with any .NET capable programming language such as: ▪ C# ▪ VB .NET ▪ J# ▪ others This interface is available in the Windows version only.
`JNI`	The Java native interface (JNI) is for use with Java.
`COM`	The component object model (COM) interface can be used with any COM-capable programming language, such as: ▪ MS Visual Basic ▪ MS Office Products such as Access or Excel (VBA) ▪ C++ ▪ VBScript ▪ others This interface is available in the Windows version only.
`C`	The native C interface is for use with C and C++.

Development

The software developer kit (SDK) contains all files that are used for developing the software. The role of each file with respect to the four different interfaces is shown in Table: Files for Development. The files are split in four categories:

Req. This file is required for this interface.

Opt. This file is optional (e.g. *Inet.dll* is used for http: and other connections. When using the API locally, this file is not used). See also Table: File Description to identify which files are required for your application.

Doc. This file is for documentation only. An empty field indicates this file is not used at all for this particular interface.

Table: Files for Development

Name	.NET	JNI	COM	C
bin\Pdf2PdfAPI.dll	**Req.**	**Req.**	**Req.**	**Req.**
bin\pdcjk.dll	Opt.	Opt.	Opt.	Opt.
bin*NET.dll	**Req.**			
bin*NET.xml	Doc.			
bin*.ocr	Opt.	Opt.	Opt.	Opt.
bin\icc*.*	Opt.	Opt.	Opt.	Opt.
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\Pdf2PdfAPI.idl			Doc.	
doc\javadoc*.*		Doc.		
include\pdf2pdfapi_c.h				**Req.**
include*.*				Opt.
jar\CNVA.jar		**Req.**		
lib\Pdf2PdfAPI.lib				**Req.**
samples*.*	Doc.	Doc.	Doc.	Doc.

The purpose of the most important distributed files of is described in Table: File Description.

Table: File Description	
Name	**Description**
`bin\Pdf2PdfAPI.dll`	This is the DLL that contains the main functionality (required).
`bin\pdcjk.dll`	This DLL contains support for Asian languages. It is loaded from the module path.
`bin\*NET.dll`	The .NET assemblies are required when using the .NET interface. The files *bin*NET.xml* contain the corresponding XML documentation for MS Studio.
`bin\*.ocr`	These are OCR interface DLLs that are used in combination with the 3-Heights™ OCR Enterprise Add-On which can be purchased as a separate product.[2]
`bin\Icc\*.ink`	Color profiles or links to websites that offer color profiles to download. Color profiles are optional. See also chapter Color Profiles.
`include\pdferror.h`	Supplementary C header file containing error codes.
`doc\*.*`	Various documentations.
`include\*.*`	Contains files to include in your C / C++ project.
`jar\CNVA.jar`	The Java wrapper.
`lib\Pdf2PdfAPI.lib`	The Object File Library needs to be linked to the C/C++ project.
`samples\*.*`	Contains sample programs in different programming languages.

Deployment

For the deployment of the software only a subset of the files are required. Which files are required (**Req.**), optional (Opt.) or not used (empty field) for the four different interfaces is shown in the table below.

Table: Files for Deployment				
Name	**.NET**	**JNI**	**COM**	**C**
`bin\Pdf2PdfAPI.dll`	**Req.**	**Req.**	**Req.**	**Req.**
`bin\pdcjk.dll`	Opt.	Opt.	Opt.	Opt.
`bin\*NET.dll`	**Req.**			
`bin\*.ocr`	Opt.	Opt.	Opt.	Opt.
`bin\icc.*`	Opt.	Opt.	Opt.	Opt.
`jar\CNVA.jar`		**Req.**		

The deployment of an application works as described below:

1. Identify the required files from your developed application (this may also include color profiles)
2. Identify all files that are required by your developed application
3. Include all these files into an installation routine such as an MSI file or simple batch script
4. Perform any interface-specific actions (e.g. registering when using the COM interface)

Example:

This is a very simple example of how a COM application written in Visual Basic 6 could be deployed.

1. The developed and compiled application consists of the file convert.exe. Color profiles are not used.
2. The application uses the COM interface and is distributed on Windows only.
   - The main DLL *Pdf2PdfAPI.dll* must be distributed.

[2] These files must reside in the same directory as Pdf2PdfAPI.dll.

- Asian text should be supported, thus *pdcjk.dll* is distributed.
- All documents used by the application have their fonts embedded (e.g. because they are compliant to PDF/A), therefore the font related files are not distributed.

3. All file are copied to the target location using a batch script. This script contains the following commands:
```
COPY Pdf2PdfAPI.dll %targetlocation%\.
COPY pdcjk.dll %targetlocation%\.
```

4. For COM, the main DLL needs to be registered in silent mode (/s) on the target system. This step requires Power-User privileges and is added to the batch script.
```
REGSVR32 /s %targetlocation%\Pdf2PdfAPI.dll
```

2.4 Uninstall, Install a New Version

If you used the MSI for the installation, go to Start ->3-Heights™ PDF to PDF/A Converter API ... ->Uninstall...
If you used the ZIP file: In order to uninstall the product undo all the steps done during installation, e.g. un-register using regsvr32 -u, delete all files, etc.

Installing a new version does not require to previously uninstall the old version. The files of the old version can directly be overwritten with the new version. If using the COM interface, the new DLL must be registered, un-registering the old version is not required.

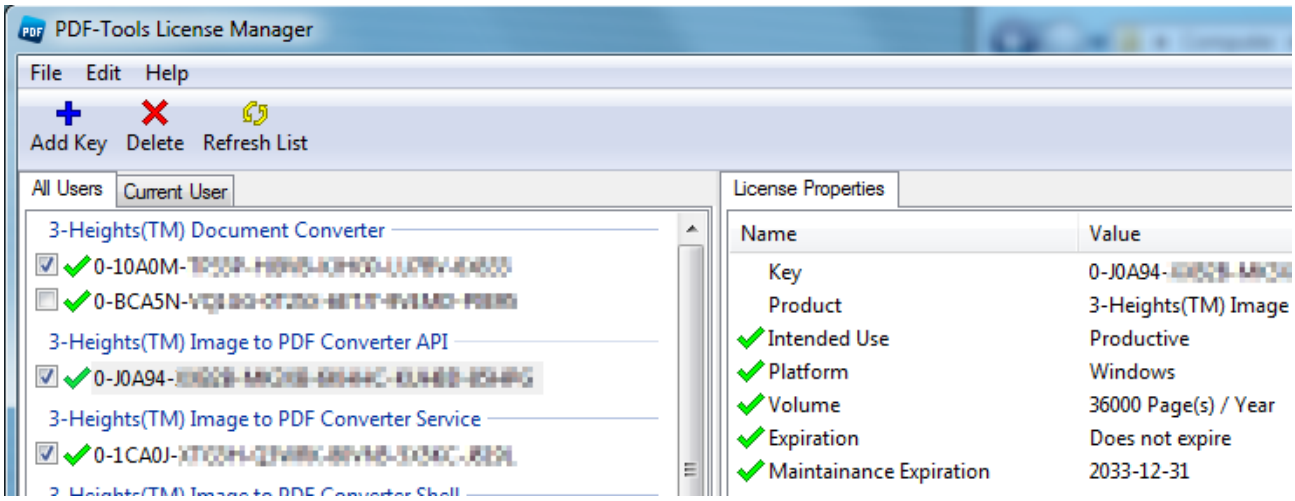
3 License Management

There are three possibilities to pass the license key to the application:

1. The license key is installed using the GUI tool (Graphical user interface). This is the easiest way if the licenses are managed manually. It is only available on Windows.
2. The license key is installed using the shell tool. This is the preferred solution for all non-Windows systems and for automated license management.
3. The license key is passed to the application at runtime via the "LicenseKey" property. This is the preferred solution for OEM scenarios.

3.1 Graphical License Manager Tool

The GUI tool *LicenseManager.exe* is located in the *bin* directory of the product kit.



List all installed license keys

The license manager always shows a list of all installed license keys in the left pane of the window. This includes licenses of other PDF Tools products. The user can choose between:

- Licenses available for all users. Administrator rights are needed for modifications.
- Licenses available for the current user only.

Add and delete license keys

License keys can be added or deleted with the “Add Key” and “Delete” buttons in the toolbar.

- The “Add key” button installs the license key into the currently selected list.
- The “Delete” button deletes the currently selected license keys.

Display the properties of a license

If a license is selected in the license list, its properties are displayed in the right pane of the window.

Select between different license keys for a single product

More than one license key can be installed for a specific product. The checkbox on the left side in the license list marks the currently active license key.

3.2 Command Line License Manager Tool

The command line license manager tool `licmgr` is available in the bin directory for all platforms except Windows.

A complete description of all commands and options can be obtained by running the program without parameters:

```
licmgr
```

List all installed license keys

```
licmgr list
```

Add and delete license keys

Install new license key:

```
licmgr store X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Delete old license key:

```
licmgr delete X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Both commands have the optional argument `-s` that defines the scope of the action:

- `g`: For all users
- `u`: Current user

Select between different license keys for a single product

```
licmgr select X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.3 License Key Storage

Depending on the platform the license management system uses different stores for the license keys.

Windows

The license keys are stored in the registry:

- `HKLM\Software\PDF Tools AG` (for all users)
- `HKCU\Software\PDF Tools AG` (for the current user)

Mac OS X

The license keys are stored in the file system:

- `/Library/Application Support/PDF Tools AG` (for all users)
- `~/Library/Application Support/PDF Tools AG` (for the current user)

Unix/Linux

The license keys are stored in the file system:

- `/etc/opt/pdf-tools` (for all users)
- `~/.pdf-tools` (for the current user)

Note: The user, group and permissions of those directories are set explicitly by the license manager tool. It may be necessary to change permissions to make the licenses readable for all users. Example:

```
chmod -R go+rx /etc/opt/pdf-tools
```

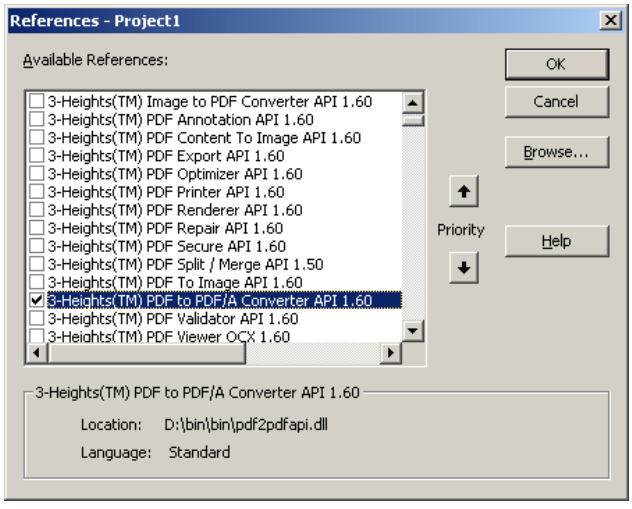
4 Programming Interfaces

4.1 Visual Basic 6

After installing the 3-Heights™ PDF to PDF/A Converter API and registering the COM interface (see chapter Download and Installation), you find a Visual Basic 6 example *Pdf2PdfAPI.vbp* in the directory *samples/VB/*. You can either use this sample as a base for an application, or you can start from scratch.

If you start from scratch, here is a quick start guide for you:

1. First create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heights™ PDF to PDF/A Converter API component to your project.


References - Project1
Available References:
OK
Cancel
Browse...
Help
Priority
3-Heights(TM) Image to PDF Converter API 1.60
3-Heights(TM) PDF Annotation API 1.60
3-Heights(TM) PDF Content To Image API 1.60
3-Heights(TM) PDF Export API 1.60
3-Heights(TM) PDF Optimizer API 1.60
3-Heights(TM) PDF Printer API 1.60
3-Heights(TM) PDF Renderer API 1.60
3-Heights(TM) PDF Repair API 1.60
3-Heights(TM) PDF Secure API 1.60
3-Heights(TM) PDF Split / Merge API 1.50
3-Heights(TM) PDF To Image API 1.60
3-Heights(TM) PDF to PDF/A Converter API 1.60
3-Heights(TM) PDF Validator API 1.60
3-Heights(TM) PDF Viewer OCX 1.60
3-Heights(TM) PDF to PDF/A Converter API 1.60
Location: D:\bin\bin\pdf2pdfapi.dll
Language: Standard

2. Draw a new Command Button and optionally rename it if you like.
3. Double-click the command button and insert the few lines of code below. All that you need to change is the path of the file name.

Example:

```
Private Sub Command1_Click()
  Dim conv As New Pdf2PdfAPI
  Dim done As Boolean
  conv.CertificateName = "Philip Renggli" ' Use "issued to" of certificate
  conv.SignatureReason = "I am the author of the document"
  conv.Compliance = ePDFA1b
  done = conv.Convert("C:\in1.pdf", "", "C:\out1.pdf", "C:\temp\log1.txt")
  Set conv = Nothing
End Sub
```

Note that applying the digital signature is optional (you may also change the certificate name), therefore the two corresponding lines could be skipped.

C Interface

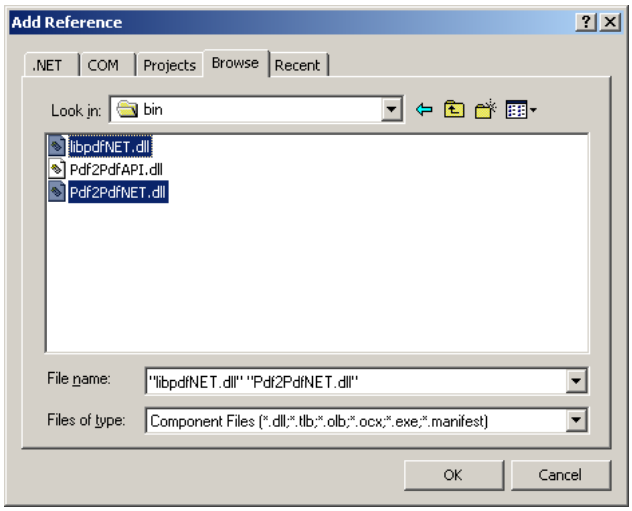
- The header file *pdf2pdfapi_c.h* needs to be included in the C/C++ program.
- The library *Pdf2PdfAPI.lib* needs to be linked to the project.
- The dynamic link library *Pdf2PdfAPI.dll* needs to be in path of executables (e.g. on the environment variable "PATH").

4.2 .NET

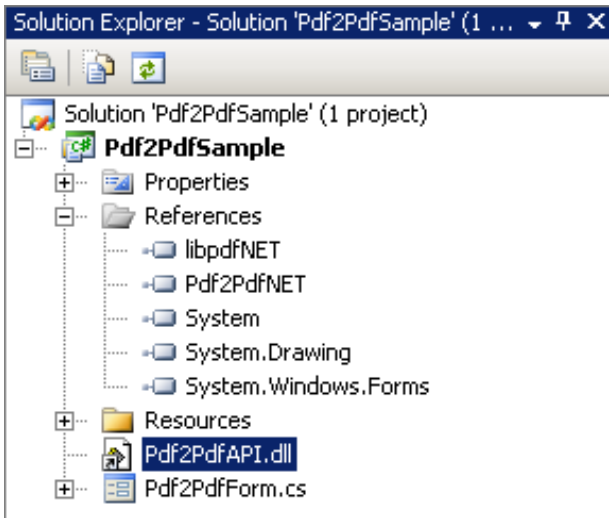
References

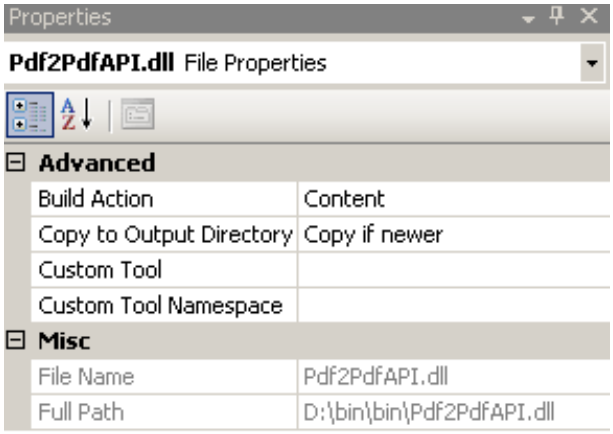
The 3-Heights™ PDF to PDF/A Converter API does not provide a pure .NET interface. Instead, it consists of .NET assemblies and a native DLL. This has to be accounted for when installing and deploying the software.

1. The .NET assemblies (**NET.dll*) are to be added as references to the project (see screenshot below).
   To do so, in the "Solution Explorer" right-click your project and select "Add Reference...". The "Add Reference" dialog will appear. In the tab "Browse", browse for the .NET assemblies *libpdfNET.dll* and *Pdf2PdfNET.dll*, add them to the project as shown below:



2. *Pdf2PdfAPI.dll* is not a .NET assembly, but a native DLL. It is not to be added as a reference to the project. (Doing so would use its COM interface and create an Interop DLL). *Pdf2PdfAPI.dll* is called by the .NET assembly *Pdf2PdfNET.dll*. *Pdf2PdfAPI.dll* must be found at execution time. The common way to do this is adding it as an existing item to the project and set its property "Copy to Output Directory" to "Copy if newer". Alternatively the directory where *Pdf2PdfAPI.dll* resides can be added to the environment variable "PATH" or it can simply be copied manually to the output directory.





   It is required to use the 32 bit version of the software on a 32 bit platform. On a 64 bit platform either version of the software can be used.
   Should you use the 32 bit version of the software on a 64 bit platform, the platform configuration must not be set to "Any CPU", but explicitly to "x86".
   The version of the .NET assemblies and the native DLL must be the same, e.g. 4.2.26.0. The version can be found in the version tag of the file's properties.

Create a New Project

There should be at least one .NET sample for MS Visual Studio 2005 available in the ZIP archive of the Windows Version of the 3-Heights™ PDF to PDF/A Converter API. The easiest for a quick start is to refer to this sample.

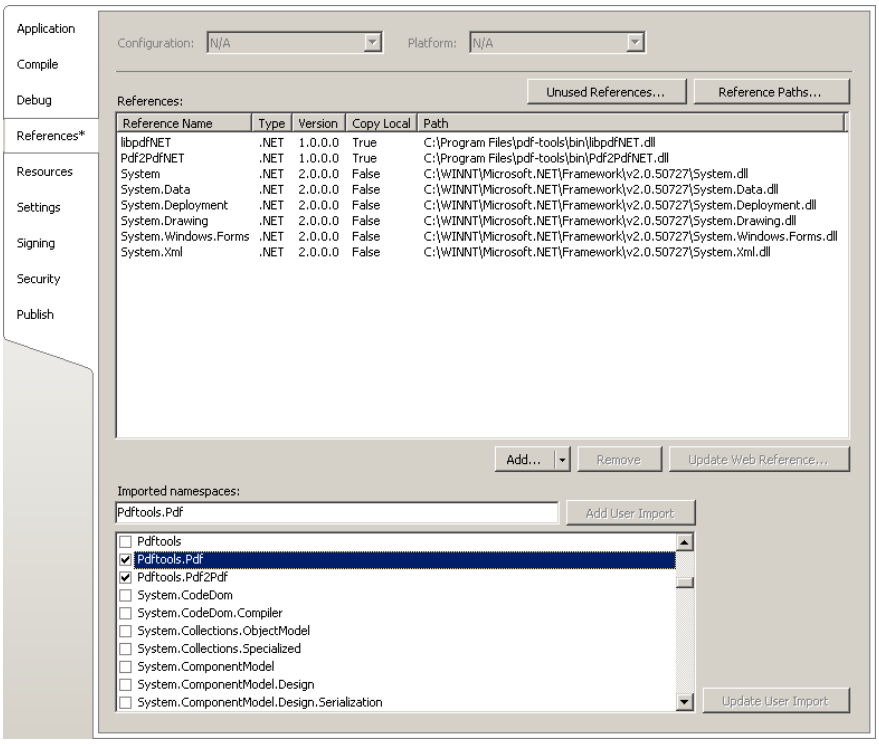
In order to create a new project from scratch, do the following steps:

1. Start Visual Studio and create a new C# or VB project.
2. Add a reference to the .NET assemblies.
3. import namespaces (Note: This step is optional, but useful.)
4. Write Code

Steps 3 and 4 are shown separately for C# and Visual Basic.

Visual Basic

3. Double-click “My Project” to view its properties. On the left hand side, select the menu “References”. The .NET assemblies you added before should show up in the upper window.
   In the lower window import the namespaces *Pdftools.Pdf* and *Pdftools.Pdf2Pdf*.
   You should now have settings similar as in the screenshot below:



4. The .NET interface can now be used as shown below:

```
Dim converter As New Pdftools.Pdf2Pdf.Pdf2Pdf()
converter.Compliance = PDFCompliance.ePDFA1b
...
converter.Convert(...)
```

C#

3. Add the following namespaces:

```
using Pdftools.Pdf;
using Pdftools.Pdf2Pdf;
```

4. The .NET interface can now be used as shown below:

```
Pdf2Pdf converter = new Pdf2Pdf();
converter.Compliance = PDFCompliance.ePDFA1b;
...
converter.Convert(...);
```

Deploying in .NET

When deploying a .NET solution, please refer to the following FAQ "Deploying in .NET":
http://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Deployment-In-NET

Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is if the native DLL is not found at execution time. This normally manifests when the constructor is called for the first time and exception is thrown - normally of type System.TypeInitializationException.

To resolve that ensure the native DLL is found at execution time. For this, see section .NET Interface in the chapter Installation or the following FAQ:
https://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Exception-type-initializer

4.3 Java Interface

For compilation and execution: When using the Java interface, the Java-wrapper *jar\CNVA.jar* needs to be on the CLASSPATH. This can be done by either adding it to the environment variable CLASSPATH, or by specifying it using the switch -classpath

```
javac -classpath .;C:\pdf-tools\jar\CNVA.jar sample.java
```

For execution: Additionally the library *Pdf2PdfAPI.dll* needs to be on the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM.

```
java -classpath .;C:\pdf-tools\jar\CNVA.jar -Djava.library.path=.;C:\pdf-tools\bin
    sample
```

4.4 ASP

The COM name of the class, for example used in ASP or PHP, of the PDF to PDF/A Converter API is PDF2PDFAPI.Pdf2Pdf.

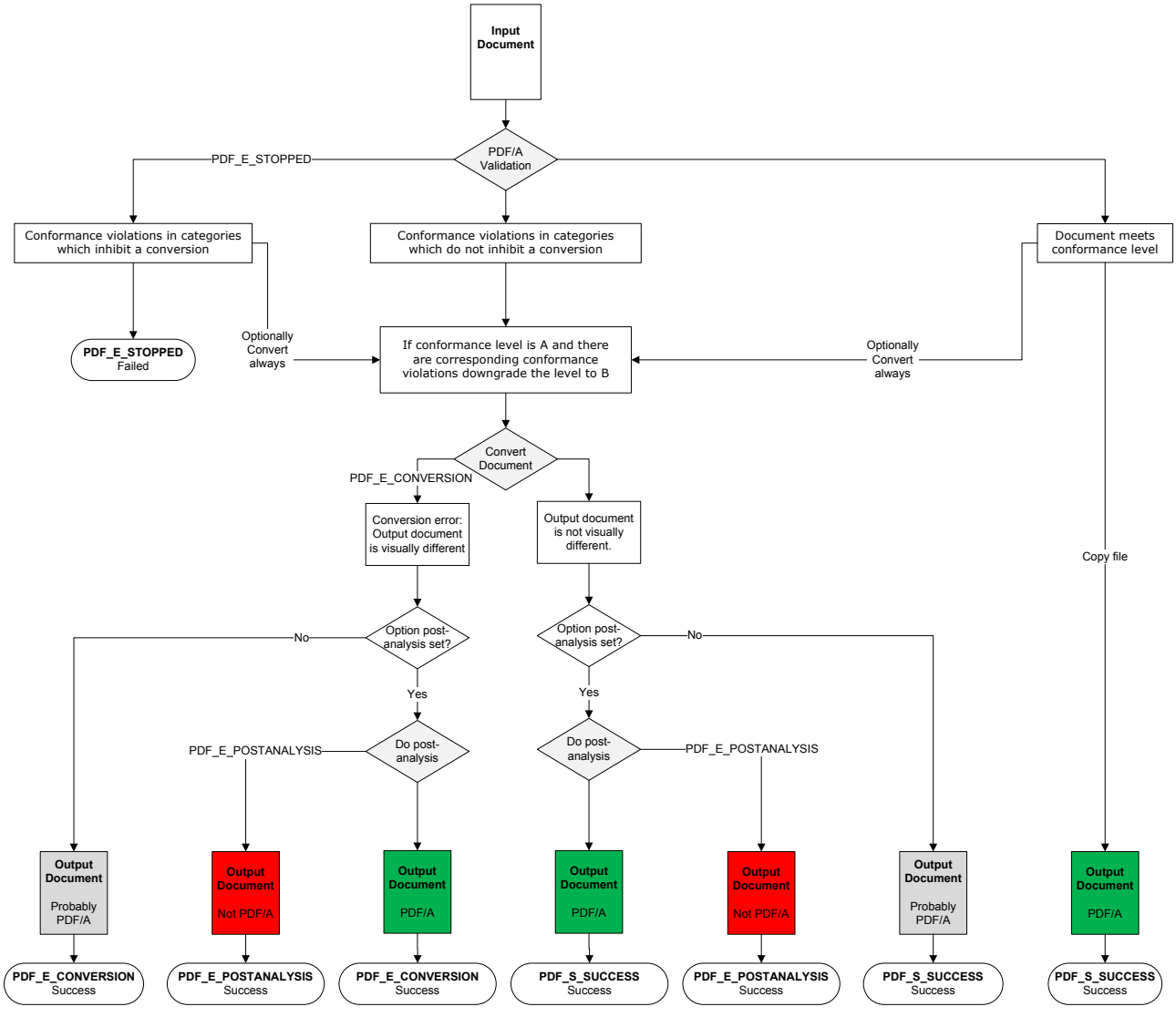
```
<%@ Language=VBScript %>
<%
  option explicit
  dim conv
  dim fileNameIn, fileNameOut, logName

  set conv = Server. CreateObject("PDF2PDFAPI.Pdf2Pdf")
  fileNameIn = "C:\PDF-Tools\doc\license.pdf"
  fileNameOut = "C:\temp\output.pdf"
  logName = "C:\temp\output.log"

  conv.PostAnalyze = True
  conv.ReportSummary = True
  if not conv.Convert(fileNameIn, "", fileNameOut, logName) then
    Response.Write "<p>"
    Response.Write "Could not create file or conversion errors. <br>"
  else
    Response.Write "<p>"
    Response.Write "Output file created successfully. <br>"
end if

Response.Write "<p>"
Response.Write "Output File (PDF/A-1b) : <a href=" & fileNameOut &
```

```
                ">" & fileNameOut & "</a><br>"
Response.Write "Log File : <a href=" & logName & ">" & logName &
                "</a><br>"
%>
```

5 User's Guide

5.1 Process Description

The workflow of the PDF to PDF/A Conversion is outlined in the graphic below.



1. *License Check*: The license is checked.
2. *Pre-Analysis*: The input document is analyzed. If the document is already compliant to the requested standard it is copied.
   If the required standard cannot be met, but a lower level of the standard can, the target level is downgraded, e.g. from PDF/A-1a to PDF/A-1b[3].
   It the target standard is PDF/A-1 and a file contains transparency or other elements that cannot be converted to PDF/A-1, the target standard is upgraded to PDF/A-2[4].

[3] Automatic downgrades can be deactivated using the property ConversionErrorMask .
[4] Automatic upgrades can be deactivated using the property ConversionErrorMask .

If the input document contains non-convertible elements the conversion is stopped, except if convert-always is enabled.

3. *Conversion*: The actual conversion is performed.
   A failure indicates a case where no output file is created. Possible causes of that are: the output file cannot be written, or conversion was aborted.
   Success indicates a correct execution of the function and creation of an output document. However it does not imply there were no problems during the process. If actions had to be taken that might have altered the visual appearance of the file or crucial data had to be removed, a conversion error is generated (see chapter "Conversion Errors" below).
4. *Post-Analysis*: If post analysis is enabled, the resulting PDF document is validated. If the resulting document does not meet the requested standard a post-analysis error is raised.

The above graphic includes PDF/A related errors only. Errors with digital signatures are not reflected in this workflow chart.

Conversion Steps

The goal of the conversion is to create a document which is conforming to the PDF/A ISO standard.

If the analysis of the document indicates a conversion to the requested standard is possible, the following steps are performed:

- Embed and subset non-embedded font programs
- Replace device specific color spaces with CIE-based color spaces
- Add a GTS_PDFA output intent
- Remove prohibited entries
- Remove entries with a default value
- Remove entries with unknown values
- Add mandatory entries
- Add XMP metadata if missing or fix inconsitent XMP metadata
- Apply implicit optimization functions (e.g. replace and subset embedded fonts)
- Apply implicit repair functions (to conform with ISO19005-1 chapter 6.1)

If the analysis indicates a conversion is not possible, a "best effort" conversion can be forced. In this case the output may or may not be PDF/A conformant. Use the post analysis feature in order to detect, whether or not the output is conformant. It is also possible that the output file looks visually different to the input file due to the forced conversion.

Conversion Errors

The conversion error (ErrorCode is PDF_E_CONVERSION) indicates that during conversion actions had to be taken that might have altered the visual appearance of the file or crucial data had to be removed. Use the property ConversionErrorMask to define what is crucial to your process and should therefore lead to a conversion error. Note that the resulting document is PDF/A compliant nonetheless.

The following issues may result in a conversion error:

- Optional content removed
- FFilter or FDecodeParms removed
- Prohibited annotation type converted to text annotation
- Prohibited action removed
- Embedded files removed
- Annotation without appearance stream
- Transparency removed
- Character from show string removed because glyph missing in font
- Unconvertible metadata

For a complete list of conversion events that can lead to a conversion error see property ConversionErrorMask.

Some of these conversion errors, such as transparency or optional content may be resolved by creating PDF/A-2 or PDF/A-3 instead of PDF/A-1.

Post Analysis

Optionally a post analysis step can be activated in order to check, whether or not the output file created conforms to the requested standard. A post analysis error (ErrorCode is PDF_E_POSTANALYSIS) indicates that the output file is not PDF/A. In case of a post analysis error you can repeat the conversion with the ReportDetails property set and look at the log file in order to see why the post analysis failed.

5.2 What is PDF/A?

PDF/A-1

The PDF/A-1 format is described in the international standard ISO-19005-1. It bases on the PDF 1.4 reference and has some additional requirements. Best is to have a general understanding of PDF/A. Here is a brief overview of how to create a PDF/A document from a non-PDF/A document.

1. A PDF/A has requirements about meta data and the structure of the file. The PDF to PDF/A Converter takes care of this and the user does not have to apply any settings. However he can provide the XMP meta data himself if desired.
2. In PDF/A, colors (including grayscale and black/white) must not be represented in a device color space (DeviceRGB, DeviceCMYK, DeviceGray). Suitable default color space profiles to substitute the device color spaces, one for RGB, CMYK and grayscale respectively can be provided by the user. In addition, or alternatively, one color space profile can be embedded as output intent. In this latter case, device colors are automatically managed by the output intent if the color can be represented in the space given by the color space profile in the output intent.
   If the converter encounters unmanaged colors, e.g. because no color space profile was set, then a calibrated color space is generated automatically, one RGB and one grayscale, for RGB and grayscale colors respectively. If unmanaged CMYK colors are encountered, a default CMYK output intent is embedded.
3. Fonts used in visible text must be embedded. This is automatically done by the Converter.
4. For PDF/A 1a: The original document structure information will be retained when converting the file to PDF/A. However, new tags will not be added and the structure will not be changed. To create a PDF/A-1a compliant file, the original file must have been created with the required structure and tagging. Otherwise, a PDF/A-1b file will be produced.

What is the difference between PDF/A-1b and PDF/A-1a?

PDF/A-1a has additional specifications on top of PDF/A-1b. These are:

1. The encoding of fonts must meet additional requirements, e.g. include a ToUnicode mapping (ISO 19005-1, chapter 6.3.8)
2. The document must contain a logical structure (ISO 19005-1, chapter 6.8)

The idea of the PDF/A-1a requirements is mainly to provide support for disabled people, i.e. by providing the required information needed for applications that support the read out loud feature.

The logical structure of the document is a description of the content of the pages. This description has to be provided by the creator of the document. It consists of a fine granular hierarchical tagging that distinguishes between the actual content and artifacts (such as page numbers, footers, layout artifacts, etc.). The tagging provides a meaningful description. Examples are “This is a Header”, “This color image shows a small sailing boat at sunset”, etc. One can easily understand this information cannot be generated automatically, it needs to be provided. This is one of the reasons why not every PDF document can be converted to PDF/A-1a.

PDF/A-2

PDF/A-2 is described in ISO 19005-2. It is based on ISO 32000-1, the standard for PDF 1.7. PDF/A-2 is meant as an extension to PDF/A-1. The second part shall complement the first part and not replace it. The most important differences between PDF/A-1 and PDF/A-2 are:

- The list of compression types has been extended by JPEG2000
- Transparent contents produced by graphic programs are allowed
- Optional contents (also known as layers) can be made visible or invisible
- Multiple PDF/A files can be bundled in one file (collection, package)
- The additional conformity level U (Unicode) allows for creating searchable files without having to fulfill the strict requirements of the conformity level A (accessibility)

Documents that contain features described above, in particular layers or transparency, should therefore be converted to PDF/A-2 rather than PDF/A-1.

PDF/A-3

PDF/A-3 is described in ISO 19005-3. It is based on ISO 32000-1, the standard for PDF 1.7. PDF/A-3 is an extension to PDF/A-2. The third part shall complement the second part and not replace it. The only two differences between PDF/A-2 and PDF/A-3 are:

- Files of any format and conformance may be embedded. Embedded files need not be suitable for long-term archiving.
- Embed files can be associated with any part of the PDF/A-3 file.

For additional information about PDF/A please visit: http://www.pdf-tools.com/pdf/pdfa-longterm-archiving-iso-19005-pdf.aspx.

5.3 Color Spaces

Colors in PDF

The PDF format supports a range of color spaces:

- *Device Color Spaces: DeviceGray, DeviceRGB, DeviceCMYK.* These are also referred to as uncalibrated color spaces, because they cannot be used to specify color values such that colors are reproducible in a predictable way on multiple output devices.
- *CIE-based Color Spaces: CalGray, CalRGB, Lab, ICCBased.* These are also referred to as device-independent color spaces, because they are inherently capable of specifying colors which can be reliably reproduced on multiple output devices.
- *Special Color Spaces: Separation and DeviceN.* These require an alternate color space from one of the previous two groups to allow the PDF consumer to simulate the color on devices which do not support the special color space.

Colors can occur in the following objects of a PDF/A document:

- Raster images (also inline images)
- Text and Vector objects such as lines and curves
- Annotations
- Shading patterns
- Transparency blending (PDF/A-2 and later)

ICC Color Profiles

An ICC (International Color Consortium) profile is a file format which can be used to describe the color characteristics of a particular device. For example for the correct color reproduction when an image from a scanner or camera is displayed on a device, such as a monitor or printer. Color profiles are usually provided with the operating system (OS), on a Windows System, they can be found at the following location:
%SystemRoot%\system32\spool\drivers\color

Alternatively, profiles can be found here:

- www.pdf-tools.com/public/downloads/resources/colorprofiles.zip
- www.color.org/srgbprofiles.html
- www.adobe.com/support/downloads/iccprofiles/icc_eula_win_dist.html

Please note that most color profiles are copyrighted, therefore you should read the license agreements on the above links before using the color profiles. The PDF to PDF/A Converter will try to locate color profiles automatically in the %SystemRoot%\system32\spool\drivers\color folder as needed. On Unix platforms, you can store the color profiles contained in the "colorprofiles.zip" download in a folder of your choice, and set the environment variable PDF_ICC_PATH to point to that folder.

PDF/A Requirements

In PDF/A the usage of uncalibrated color spaces (DeviceGray, DeviceRGB, and DeviceCMYK) is prohibited because colors that are specified in this way cannot be reproduced reliably on multiple output devices. Therefore, when converting to PDF/A, all device color spaces should be replaced by CIE-based color spaces. There is one exception to this rule: An uncalibrated color is tolerated if the output intent holds an ICC color profile with which this color can be represented. (E.g. a grayscale color can be represented in an RGB color profile, but a CMYK color cannot.)

The 3-Heights™ PDF to PDF/A Converter API uses the following strategy:

- For each device color space (DeviceGray, DeviceRGB, and DeviceCMYK) an ICC color profile can be specified to be used as substitute for the respective device color space.
- One ICC color profile can be set to be used in the output intent.
- During conversion, if a device color space is encountered then the following is done:
  - If an output intent was set that is capable of managing this color, no action is needed.
  - Otherwise, if an ICC color profile is set to substitute this device color space then this color profile is used.
  - Otherwise, for DeviceRGB and DeviceGray color spaces: A calibrated color space (CalRGB[5] and CalGray respectively) is generated and used as a substitute.
  - Otherwise, for DeviceCMYK color spaces:
    - If the output intent is not set, then a default CMYK ICC color profile is used for the output intent.
    - If the output intent holds a non-CMYK ICC color profile, then a default CMYK ICC color profile is generated and used as a substitute for DeviceCMYK.

The above strategy is motivated by the fact that CalRGB and CalGray color spaces occupy very little memory in comparison to ICC color profiles. Also note that the primary purpose of the output intent in a PDF document is to describe the characteristics of the device on which a document is intended to be rendered. Traditionally, the target device is a printer, which motivates CMYK output intents. The default CMYK color profile *USWebCoatedSWOP.icc* is provided in the sub-directory *bin\icc*.

[5]The generated CalRGB color space is an approximation to the ICC color profile *sRGB Color Space Profile.icm*.

5.4 Fonts

The PDF/A standard requires all fonts to be embedded in the PDF file. This ensures that the future rendering of the textual content of a conforming file matches, on a glyph by glyph basis, the appearance of the file as originally created.

Hence, if non-embedded fonts in a PDF are used, the font must be embedded. For this, a matching font has to be found in the font directories. The method `AddFontDirectory` should be used to define your font directories. The default font directories are listed in the documentation to the method `AddFontDirectory`.

It is important that the font directories contain all fonts that are used for the input files. In particular, the font ZapfDingbats is widely used in PDF documents, but not available on most systems. Therefore, the product kit includes this font, which should be added to a font directory.

Fonts should be added to one of the font directories, if the post analysis returns validation errors like the following: `"output.pdf", 9, 20, 0x00418704, "The font ShinGo must be embedded.", 1`

The fonts of the font directories and their properties are cached in a font cache, located in the files `font-database*` in the temporary files folder. The cache files have to be removed manually, if fonts are added or removed from these directories.

The directory for temporary files is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

- *Windows:*
  1. The path specified by the *TMP* environment variable.
  2. The path specified by the *TEMP* environment variable.
  3. The path specified by the *USERPROFILE* environment variable.
  4. The Windows directory.
- *Unix:*
  1. The path specified by the *PDFTMPDIR* environment variable.
  2. The path specified by the *TMP* environment variable.
  3. The */tmp* directory.

Example: Command to remove the font cache files on Windows

```
del %TMP%\font-database*
```

Microsoft Core Fonts on Unix

Many PDF documents use Microsoft core fonts like Arial, Times New Roman and other fonts commonly used on Windows. Therefore, it is recommended to install these fonts to your default font directories. Many Linux distributions offer an installable package for these “Microsoft TrueType core fonts”. For instance, on Debian based systems the package is called `ttf-mscorefonts-installer`.

Alternatively you can download the fonts from here:
http://corefonts.sourceforge.net/

Microsoft has an FAQ on the subject, that covers licensing related questions as well:
http://www.microsoft.com/typography/faq/faq8.htm

Font Configuration File fonts.ini

The font configuration file can be used to control the embedding of fonts. The file fonts.ini must reside at the following location, which is platform dependent:

- *Windows:* In a directory named “Fonts”, which must be a direct sub-directory of where the main DLL or executable resides.
- *Unix:* The “fonts.ini” file is searched in the following locations
  1. If the environment variable PDFFONTDIR is defined: `$PDFFONTDIR/fonts.ini`
  2. `~/.pdf-tools/fonts/fonts.ini`
  3. `/etc/opt/pdf-tools/fonts/fonts.ini`

fonts.ini uses the ini file format and has two sections. The section "fonts" is ignored by the PDF to PDF/A Converter, so you may remove it. In the section "replace" font replacement rules of the form key=value can be defined. The key specifies the font that is to be replaced. The key should match the name of the font mentioned in the pre-analysis of the PDF to PDF/A Converter. e.g. "ShingGo" for:

```
"file.pdf", 9, 20, 0x00418704, "The font ShinGo must be embedded.", 1
```

The value should match the true type name of an installed font. Do not replace any standard fonts (Helvetica, Arial, Times, TimesNewRoman, Courier, CourierNew, Symbol, and ZapfDingbats).

Please note that this feature should be used with care. Replacing a font with another might change the visual appearance of the file because of different glyph shapes or glyphs that are not available in the replacement font. Embedding another font might also have legal consequences.

Example: Replace MS-Mincyo with MS-Mincho

```
[replace]
MS-Mincyo=MS-Mincho
```

This rule defines, that in order to embed a font program for font MS-Mincyo the font MS-Mincho should be used. This rule is useful, because both names are possible transliterations of the same Japanese font. However, the official transliteration used by the actual font is MS-Mincho.

5.5 Cryptographic Provider

In order to use the 3-Heights™ PDF to PDF/A Converter API's cryptographic functions such as creating digital signatures, a cryptographic provider is required. The cryptographic provider manages certificates, their private keys and implements cryptographic algorithms.

The 3-Heights™ PDF to PDF/A Converter API can use various different cryptographic providers. The following list shows, for which type of signing certificate which provider can be used.

USB Token or Smart Card

These devices typically offer a PKCS#11 interface, which is the recommended way to use the certificate → PKCS#11 Provider.

On Windows, the certificate is usually also available in the Microsoft CryptoAPI. This provider is not recommended, unless you experience problems with your device's PKCS#11 interface.

If you need to sign documents on a non-Windows system with an USB token that does not come with middleware for your platform, you can use the 3-Heights™ Signature Creation and Validation Service.

If you need to sign documents on Windows in a non-interactive or locked session[6], use the 3-Heights™ Signature Creation and Validation Service.

Hardware Security Module (HSM)

HSMs always offer very good PKCS#11 support → PKCS#11 Provider

For more information and installation instructions see separate document TechNotePKCS11.pdf.

Soft Certificate

Soft certificates are typically PKCS#12 files that have the extension `.pfx` or `.p12` and contain the signing certificate as well as the private key and trust chain (issuer certificates). Soft certificate files cannot be used directly. Instead, they must be imported into the certificate store of a cryptographic provider.

- *All Platforms*: The recommended way of using soft certificates is to import them into a store that offers a PKCS#11 interface and use the PKCS#11 Provider. For example:
  - A HSM
  - openCryptoki on Linux
  - PKCS#11 softtoken on Solaris

  For more information and installation instructions of the above stores see separate document TechNotePKCS11.pdf.
- *Windows*: If no PKCS#11 provider is available, soft certificates can be imported into Windows certificate store, which can then be used as cryptographic provider → Microsoft CryptoAPI

[6] See the description of the 3-Heights™ Signature Creation and Validation Service for more details on this topic.

Signature Service

Signature services are a convenient alternative to storing certificates and key material locally. The 3-Heights™ PDF to PDF/A Converter API can use various different services whose configuration is explained in the following sections of this documentation:

- 3-Heights™ Signature Creation and Validation Service
- SwissSign Personal Signing Service
- SwissSign SuisseID Signing Service
- QuoVadis sealsign
- Swisscom All-in Signing Service

5.5.1 PKCS#11 Provider

PKCS#11 is a standard interface offered by most cryptographic devices such as HSMs, USB Tokens or sometimes even soft stores (e.g. openCryptoki).

More information on and installation instructions of the PKCS#11 provider of various cryptographic devices can be found in the separate document TechNotePKCS11.pdf.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

```
"PathToDll;SlotId;Pin"
```

- `PathToDll` is the path to driver library filename, which is provided by the manufacturer of the HSM, UBS token or smart card. Examples:
  - The SuisseID USB Tokens use *cvp11.dll*
  - The CardOS API from Atos (Siemens) uses *siecap11.dll*
  - The IBM 4758 cryptographic coprocessor uses *cryptoki.dll*
  - Devices from Aladdin Ltd. use *etpkcs11.dll*
- `SlotId` is optional, if it is not defined, it is searched for the first slot that contains a running token.
- `Pin` is optional, if it is not defined, the submission for the pin is activated via the pad of the token. If this is not supported by the token, the following error message is raised when signing: “Cannot access private key”.

Examples:

- `Provider = "\WINDOWS\system32\siecap11.dll;4;123456"`

Note: Some PKCS#11 drivers require the Terminate method to be called. Otherwise your application might crash upon termination.

The chapter Guidelines for Mass Signing contains important information to optimize performance when signing multiple documents.

Interoperability Support

The following cryptographic token interface (PKCS#11) products have been successfully tested:

- SafeNet Protect Server
- SafeNet Luna
- SafeNetAuthentication Client
- IBM OpenCrypTokI
- CryptoVision
- Siemens CardOS

Selecting a Certificate for Signing

The 3-Heights™ PDF to PDF/A Converter API offers different ways to select a certificate. The product tries the first of the following selection strategies, for which the required values have been specified by the user.

1. **Certificate fingerprint**
   Property SignerFingerprint

- SHA1 fingerprint of the certificate. The fingerprint is 20 bytes long and can be specified in hexadecimal string representation, e.g. "b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5". In Windows certificate store this is called "Thumbprint", if "Thumbprint algorithm" is "sha1".

2. **Certificate Issuer and SerialNumber**
   Properties Issuer and SignatureSerialNumber
   - Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".
   - Serial number of the certificate (hexadecimal string representation, e.g. "4c 05 58 fb"). This is a unique number assigned to the certificate by its issuer. In Windows certificate store this is the field called "Serial number" in the certificate's "Details" tab.
3. **Certificate Name and optionally Issuer**
   Properties CertificateName and Issuer
   - Common Name of the certificate (e.g. "PDF Tools AG"), in Windows certificate store this is called "Issued To".
   - Optional: Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".

Using PKCS#11 stores with missing issuer certificates

Some PKCS#11 devices contain the signing certificate only. However, in order to embed revocation information it is important, that the issuer certificates, i.e. the whole trust chain, is available as well.

On Windows, missing issuer certificates can be loaded from the Windows certificate store. So the missing certificates can be installed as follows:

1. Get the certificates of the trust chain. You can download them from the website of your certificate provider or do the following:
   (a) Sign a document and open the output in Adobe Acrobat
   (b) Go to "Signature Properties" and then view the signer's certificate
   (c) Select a certificate of the trust chain
   (d) Export the certificate as "Certificate File" (extension .cer)
   (e) Do this for all certificates of the trust chain
2. Open the exported files by double clicking on them in the Windows Explorer
3. Click button "Install Certificate..."
4. Select "automatically select the certificate store based on the type of certificate" and finish import

5.5.2 Microsoft CryptoAPI Provider

Microsoft CryptoAPI (MS-CAPI, CAPI) offers access to the certificates stored in the Windows certificate store and other devices, such as USB tokens, with Windows integration. Microsoft CryptoAPI does not support some new cryptographic algorithms. Therefore it is recommended to use the PKCS#11 Provider if possible.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

```
"[ProviderType:]Provider[;PIN]"
```

The ProviderType and PIN are optional. The corresponding drivers must be installed on Windows.
Examples:

- `Provider = "Microsoft Base Cryptographic Provider v1.0"`
- `Provider = "Microsoft Strong Cryptographic Provider"`
- `Provider = "PROV_RSA_AES:Microsoft Enhanced RSA and AES Cryptographic Provider"`

The provider type PROV_RSA_AES supports the SHA-2 hash algorithms for signature validation (not signature creation). This provider type is recommended in order to validate signatures if no PKCS#11 device is available.
Optionally, when using an advanced certificate, the pin code can be passed as an additional, semi-column separated parameter. This does not work with qualified certificates, because they always require the pin code to be entered manually and every time. If the name of the provider is omitted, the default provider is used.
Examples, "123456" being the pin code:

- `Provider = "Microsoft Base Cryptographic Provider v1.0;123456"`
- `Provider = ";123456"`

Certificate Store Property Store

The value for the certificate store depends on the OS. Supported values are: “CA”, “MY” and “ROOT”. For signature creation the default store “MY” is usually the right choice.

Store Location Property StoreLocation

Either of the following store locations

- Local Machine
- Current User (default)

Usually personal certificates are stored in the current user location and company-wide certificates are stored under local machine.

The current user’s store is only available, if the user profile has been loaded. This may not be the case in certain environments such as within an IIS web application or COM+ applications. Use the store of the Local Machine, if the user profile cannot be loaded.

Certificates in the store Local Machine are available to all users. However, in order to sign a document, you need access to the signing certificate’s private key. The private key is protected by Windows ACLs and typically readable for Administrators only. Use the Microsoft Management Console (mmc.exe) in order to grant access to the private key for other users as follows: Add the Certificates Snap-in for the certificates on Local Machine. Right-click on the signing certificate, click on “All Tasks” and then “Manage Private Keys...” where you can set the permissions.

Selecting a Certificate for Signing

First, the certificate store defined by the provider is used. Within the store the selection of the signing certificate works the same as with the PKCS#11 provider, which is described here: Selecting a Certificate for Signing

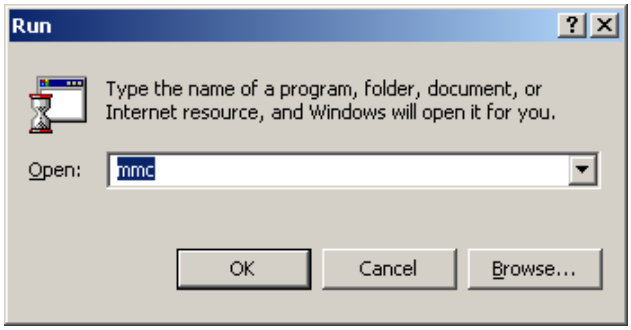
Certificates

In order to sign a PDF document, a valid, existing certificate name must be provided and its private key must be available.

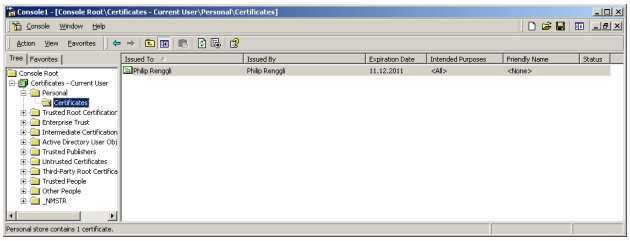
There are various ways to create or obtain a certificate. How this is done is not described in this document. This document describes the requirements for, and how to use the certificate.

On the Windows operating system certificates can be listed by the Microsoft Management Console (MMC), which is provided by Windows. In order to see the certificates available on the system, do the following steps:

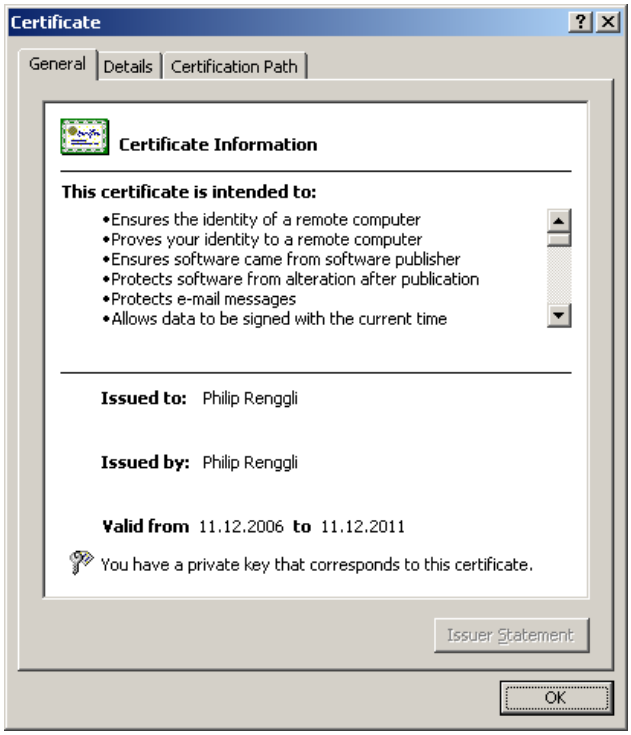
1. To launch the MMC, go to Start ->Run...->type “mmc”, or start a Command Prompt and type “mmc”.



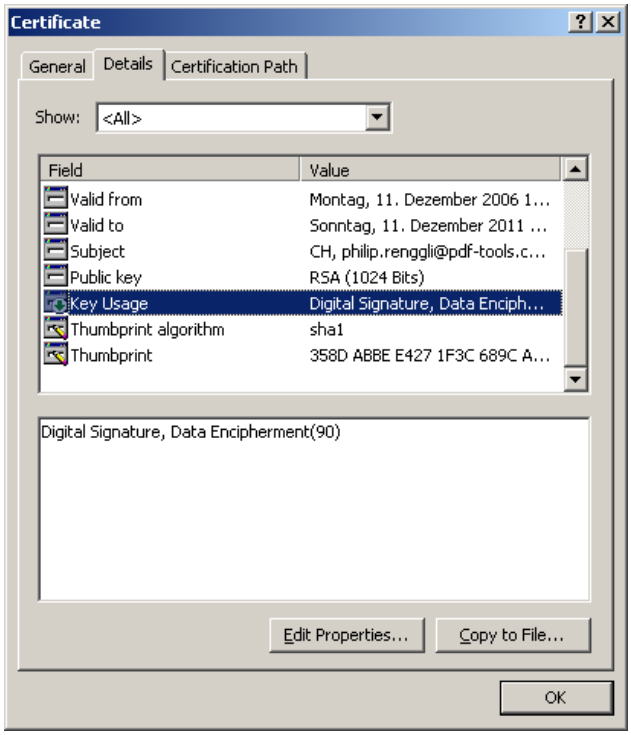
2. Under “File” ->“Add/Remove Snap-in”
3. Choose “Certificates” and click the “Add” button
4. In the next window choose to manage certificates for “My user account”
5. Click “Finish”
6. The certificate must be listed under the root “Certificates - Current User”, for example as shown in the screenshot below:



7. Double-click the certificate to open. The certificate name corresponds to the value “Issued to:”.

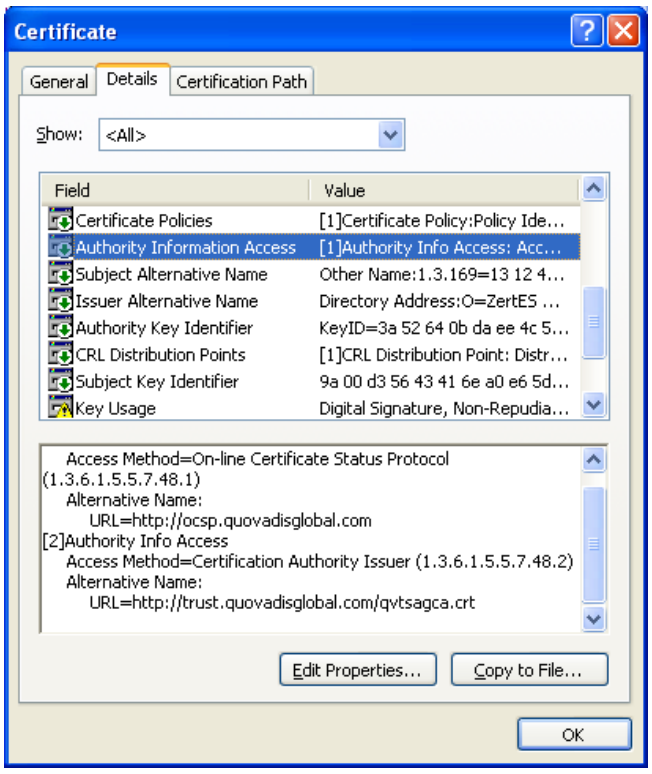

Certificate
General
Details
Certification Path
Certificate Information
This certificate is intended to:
•Ensures the identity of a remote computer
•Proves your identity to a remote computer
•Ensures software came from software publisher
•Protects software from alteration after publication
•Protects e-mail messages
•Allows data to be signed with the current time
Issued to: Philip Renggli
Issued by: Philip Renggli
Valid from 11.12.2006 to 11.12.2011
You have a private key that corresponds to this certificate.
Issuer Statement
OK

8. In the tab Detail of the certificate, there is a field named “Key Usage”. This field must contain the value “Digital Signature”. Additional values are optional, see also screenshot. You must have the private key that corresponds to this certificate.



Qualified Certificates

A qualified certificate can be obtained from a certificate authority (CA). Besides the requirements listed in the previous chapter it has the additional requirement to contain the key “Authority Information Access” which contains the information about the OCSP server.



5.5.3 3-Heights™ Signature Creation and Validation Service

The 3-Heights™ Signature Creation and Validation Service provides HTTP protocol based remote access to cryptographic providers such as smartcards, USB tokens, and other cryptographic infrastructure such as HSMs.

Use of the 3-Heights™ Signature Creation and Validation Service provides the following advantages:

1. By means of this service the tokens can be hosted centrally and used by any client computer which has access to the service.
2. Cryptographic devices that can be used on Windows only can be made accessible to siging processes running on Non-Windows systems.
3. Cryptographic devices can be made accessible to processes running in non-interactive sessions. Many cryptographic devices must always be used in an interactive session for two reasons. First, the middleware requires the user to enter the pin interactively to create a qualified electronic signature. Second, USB tokens and smart cards are managed by Windows such that the device is available only to the user currently using the computer’s console. Therefore, services, remotely logged in users and applications running in locked sessions have no access to the device.

Note that this is a separate product and this chapter describes its usage with the 3-Heights™ PDF to PDF/A Converter API only. For more information on the 3-Heights™ Signature Creation and Validation Service and installation instructions, please refer to its separate user manual.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

`"http://server.mydomain.com:<port>/<token>;<password>"`

Where:

- `server.mydomain.com` is the hostname of the server
- `<port>` is optional, port of the server.
- `<token>` the ID of the token.

- `<password>` password of the token.

Examples:

- `Provider = "http://server.mydomain.com:8080/0001;pass01"`

A more detailed description can be found in the user manual of the 3-Heights™ Signature Creation and Validation Service.

Selecting a Certificate for Signing

Selection of the signing certificate works the same as if the token was used directly: Selecting a Certificate for Signing.

5.5.4 SwissSign Personal Signing Service

Provider Property Provider

The provider configuration string contains the URL to the service endpoint.

Provider Configuration

The provider can be configured using provider session properties.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
Identity	String	required	The identity of your signing certificate. Example: `My Company:Signing Cert 1`
DSSProfile	String	required	`http://dss.swisssign.net/dss/profile/pades/1.0`
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!
RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like `AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F`

Signature Configuration

The signature can be customized using standard properties of the 3-Heights™ PDF to PDF/A Converter API.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[7].	Property CertificateName.
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

[7] This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

5.5.5 SwissSign SuisseID Signing Service

In order to use the SuisseID Signing Service, please contact Swiss Post Solutions AG (suisseid@post.ch) to obtain access credentials. Prior to invoking the SuisseID Signing Service, user authentication via the SuisseID Identity Provider (IDP) is a pre-requisite. So the calling application must integrate via SAML (e.g. SuisseID SDK) with the SuisseID Identity Provider. The IDP issues SAML tokens upon successful user authentication.

Note that the name of the signature should be the signer's name (e.g. "<givenname> <surname>"). The signer's name can be retrieved for the SAML token as the IDP provides this as qualified attributes (yellowid verified).

Provider Property Provider

The provider configuration string contains the URL to the service Endpoint.

Provider Configuration

The provider can be configured using provider session properties.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
SAMLToken	File	required	SAML token issued by the SuisseID Identity Provider (IDP). Example: `C:\temp\my-saml.xml` Note: The SAML token received from the IDP is a signed XML and must not be modified in any way. For example, the token should not be read into a string or XML object and re-serialized.
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!

Signature Configuration

The signature can be customized using standard properties.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[8].	Property CertificateName.
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

[8]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

5.5.6 QuoVadis sealsign

Provider Property Provider

The provider configuration string contains the URL to the QuoVadis sealsign service. For example:
`"http://services.sealsignportal.com:18080/sealsign/ws/BrokerClient"`

Provider Configuration

The provider can be configured using provider session properties that can be set using the method SetSessionPropertyString.

Name	Type	Required	Value
Identity	String	required	The account ID is the unique name of the account specified on the server. Example: `Rigora`
Profile	String	required	The profile identifies the signature specifications by a unique name. Example: `Default`
secret	String	required	The secret is the password which secures the access to the account. Example: `NeE=EKEd33FeCk70`
clientId	String	optional	A client ID can be used to help separating access and creating better statistics. If specified in the account configuration it is necessary to provide this value. Example: `3949-4929-3179-2818`
pin	String	required	The PIN code is required to activate the signing key. Example: `123456`
MessageDigestAlgorithm	String	optional	The message digest algorithm to use. Note that the supported algorithms depend on the provider. Default: `SHA-256` Alternatives: `SHA-1`, `SHA-384`, `SHA-512`, `RIPEMD-160`, `RIPEMD-256`

Signature Configuration

The signature can be customized using standard properties.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[9].	Property CertificateName.
Time-stamp	-	Not available.	
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

[9]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

5.5.7 Swisscom All-in Signing Service

General Properties

To use the signature service, the following general properties have to be set:

Description	Required	Value	Setting
Common Name	required	Name of the signer[10].	Property CertificateName
Provider	required	The service endpoint URL of the REST service. Example: `https://ais.swisscom.com/AIS-Server/rs/v1.0/sign`	Property Provider
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses	Property EmbedRevocationInfo

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

Provider Session Properties

In addition to the general properties, a few provider specific session properties have to be set.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
DSSProfile	String	required	`http://ais.swisscom.ch/1.0`
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!
Identity	String	required	The identity string as provided by Swisscom: `<customer name>:<key identity>`
RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like `AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F`

[10]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

On-Demand Certificates

To request an on-demand certificate, the following additional property has to be set:

Name	Type	Required	Value
SwisscomAllInOnDemandDN	String	required	The requested distinguished name. Example: `cn=Hans Muster,o=ACME,c=CH`

Step-Up Authorization using Mobile-ID

To use the step-up authorization, the following additional properties have to be set:

Name	Type	Required	Value
SwisscomAllInMSISDN	String	required	Mobile phone number. Example: `+41798765432`
SwisscomAllInMessage	String	required	The message to be displayed on the mobile phone. Example: `Pipapo halolu.`
SwisscomAllInLanguage	String	required	The language of the message. Example: `DE`

Those properties have to comply with the Swisscom Mobile-ID specification.

5.6 How to Create Digital Signatures

This chapter describes the steps that are required to create different types of digital signatures. A good introductory example can be found in the chapter Digital Signatures.

5.6.1 How to Create a PAdES LTV Signature

In order to create a PAdES LTV signature, the following is required:

1. *An advanced or qualified signing certificate.*
   For requirements and preparation steps see the sample in chapter Digital Signatures. Make sure the store of your cryptographic provider contains all certificates of the trust chain, including the root certificate.
2. *Embed revocation information.*
   Do not set the property EmbedRevocationInfo to false.
3. *Add a Time-stamp.*
   Use the property TimeStampURL.
4. *Proper error handling.*
   A proper error handling is crucial in order to ensure the creation of correctly signed documents. The output document was signed successfully, if and only if the method Convert returns true and neither of the conversion events ePDFConversionErrorSigCrea or ePDFConversionErrorNetwork occur.

5.6.2 How to Create a Visual Appearance of a Signature

Each signature may have a visual appearance on a page of the document. The visual appearance is optional and has no effect on the validity of the signature. Because of this and because a visual appearance may cover important content of the page, many applications choose to create an invisible signature.

By default, the 3-Heights™ PDF to PDF/A Converter API creates an appearance in the lower left corner of the last page, which looks as shown below:

How to Create an Invisible Signature

Invisible signatures have no visual appearance and can be created by setting an empty rectangle using the Method SetSignatureRectangle:

```
conv.SetSignatureRectangle(0, 0, 0, 0)
```

How to Create a Visual Appearance

Different properties of the visual appearance can be specified.

Page and Position
: See methods SignaturePageNo and SetSignatureRectangle.

Background image
: See propery SignatureImageFileName.

5.6.3 Guidelines for Mass Signing

This section provides some guidelines for mass signing using the 3-Heights™ PDF to PDF/A Converter API.

Keep the session to the security device open for multiple sign operations

Creating and ending the session to the security device is a complex operation. By re-using the session for multiple sign operations, performance can be improved:

1. Create a `Pdf2PdfAPI` object
2. Open the session to the provider using `BeginSession`
3. Use the Pdf2PdfAPIobject to sign multiple documents
4. Close the session to the provider using `EndSession`
5. Dispose of the `Pdf2PdfAPI` object

Signing concurrently using multiple threads

The 3-Heights™ PDF to PDF/A Converter API is thread-safe. Each `Pdf2PdfAPI` object should be used in one thread at the time only. It is recommended that each thread has a separate `Pdf2PdfAPI` object.

The performance improvement when signing concurrently using multiple threads depends mainly on the security device used. Typically the improvement is large for HSMs and small for USB Tokens.

Thread safety with a PKCS#11 provider

The PKCS#11 standard specifies, that "an application can specify that it will be accessing the library concurrently from multiple threads, and the library must [...] ensure proper thread-safe behavior." However, some PKCS#11 provider (middleware) implementations are not thread-safe. For this reason, the 3-Heights™ PDF to PDF/A Converter API synchronizes all access to the same provider (middleware and slot id).

If your middleware is thread-safe, you can enable full parallel usage of the cryptographic device by setting the session property `"LOCKING_OK"` to the value `"true"` using the method `SetSessionPropertyString`:

Example: Enable parallel access to the cryptographic device.

```
doc.SetSessionPropertyString("LOCKING_OK", "true");
```

5.6.4 Miscellaneous

Caching of CRLs, OCSP and TSP Reponses

In order to improve the speed when mass signing, the 3-Heights™ PDF to PDF/A Converter API provides a caching algorithm to store CRL (Certificate Revocation List), OCSP (Online Certificate Status Protocol), TSP (Time-stamp Protocol) and data from signature services. This data is usually valid over period of time that is defined by the protocol, which is normally at least 24 hours. Caching improves the speed, because there are situations when the server does not need to be contacted for every digital signature. The following caches are stored automatically by the 3-Heights™ PDF to PDF/A Converter API at the indicated locations within the directory for temporary files:

OCSP responses: *ocsp/server-hash.der*

CRL: *crl/server.der*

TSP responses[11] *tsp/server.der*

Service data: *sig/hash.bin*

The caches can be cleared by deleting the files. Usage of the caches can be deactivated by setting the NoCache flag. The files are updated if the current date and time exceeds the "next update" field in the OCSP or CRL response respectively or the cached data was downloaded more than 24 hours ago.

The directory for temporary files is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

- *Windows:*
  1. The path specified by the *TMP* environment variable.
  2. The path specified by the *TEMP* environment variable.
  3. The path specified by the *USERPROFILE* environment variable.
  4. The Windows directory.
- *Unix:*
  1. The path specified by the *PDFTMPDIR* environment variable.
  2. The path specified by the *TMP* environment variable.
  3. The */tmp* directory.

How to Use a Proxy

The 3-Heights™ PDF to PDF/A Converter API can use a proxy server for all communication to remote servers, e.g. to download CRL or for communication to a signature service. The proxy server can be configured using the provider session property *Proxy*. The property's value must be a string with the following syntax:

```
http[s]://<user:password>@host:<port>
```

Where:

- `http` / `https`: Protocol for connection to proxy.
- `user:password` (optional): Credentials for connection to proxy (basic authorization).
- `host`: Hostname of proxy.
- `port`: Port for connection to proxy.

For SSL connections, e.g. to a signature service, the proxy must allow the HTTP CONNECT request to the signature service.

Example: Configuration of a proxy server that is called `myproxy` and accepts HTTP connections on port `8080`.

```
conv.SetSessionPropertyString "Proxy" "http://myproxy:8080"
```

[11] The sizes of the TSP responses are cached only. Cached TSP responses cannot be embedded but used for the computation of the signature length only.

Configuration of Proxy Server and Firewall

For the application of a Time-stamp or online verification of certificates, the signature software requires access to the server of the certificates' issuer (e. g. http://ocsp.quovadisglobal.com or http://platinum-qualified-g2.ocsp.swisssign.net/) via HTTP. The URL for verification is stored in the certificate; the URL for Time-stamp services is provided by the issuer. In case these functions are not configured, no access is required.

In organizations where a web proxy is in used, it must be ensured that the required MIME types are supported. These are:

OCSP

- `application/ocsp-request`
- `application/ocsp-response`

Time-stamp

- `application/timestamp-query`
- `application/timestamp-reply`

Signature services

- Signature service specific MIME types.

5.7 How to Validate Digital Signatures

Validation of a Qualified Electronic Signature

There are basically three items that need to be validated:

1. Trust Chain
2. Revocation Information (optional)
3. Time-stamp (optional)

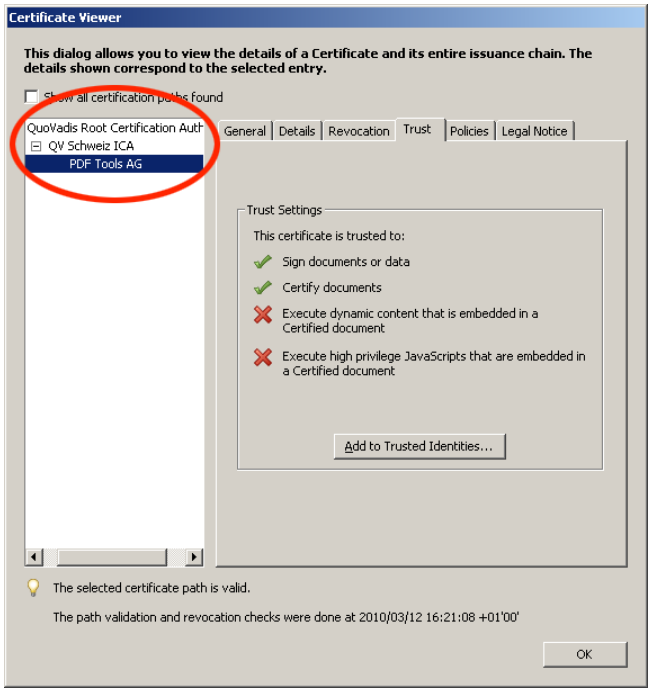
Validation can be in different ways, e.g. Adobe Acrobat, from which the screenshots below are taken.

Trust Chain

Before the trust chain can be validated, ensure the root certificate is trusted. There are different ways to add a certificate as trusted root certificate. The best way on Windows is this:

1. Retrieve a copy of the certificate containing a public key. This can be done be requesting it from the issuer (your CA) or by exporting it from an existing signature to a file (*CertExchange.cer*). Ensure you are not installing a malicious certificate!
2. Add the certificate to the trusted root certificates. If you have the certificate available as file, you can simply double-click it to install it.

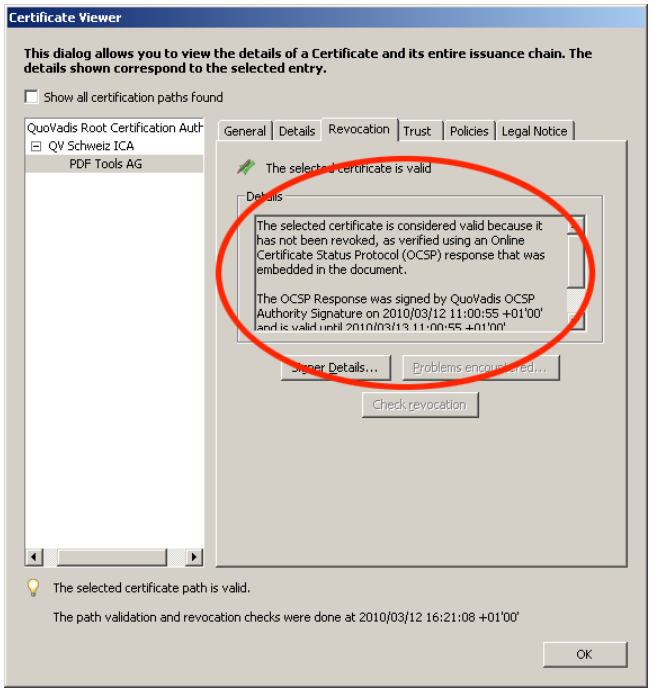
After that you can validate the signature, e.g. by open the PDF document in Adobe Acrobat, right-click the signature and select “Validate”, then select “Properties” and select the tab “Trust”. There the certificate should be trusted to “sign documents or data”.


Certificate Viewer
This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.
Show all certification paths found
QuoVadis Root Certification Auth
QV Schweiz ICA
PDF Tools AG
General
Details
Revocation
Trust
Policies
Legal Notice
Trust Settings
This certificate is trusted to:
Sign documents or data
Certify documents
Execute dynamic content that is embedded in a Certified document
Execute high privilege JavaScripts that are embedded in a Certified document
Add to Trusted Identities...
The selected certificate path is valid.
The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'
OK

Revocation Information

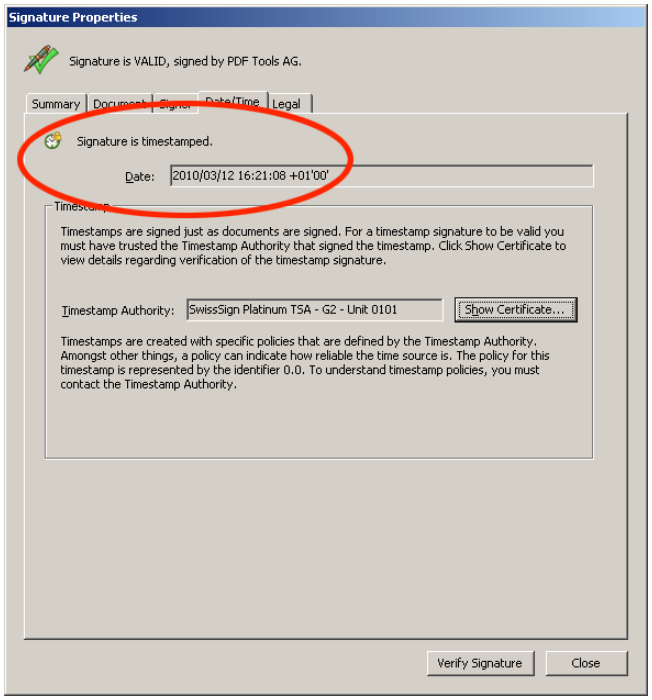
An OCSP response or CRL must be available. This is shown in the tab “Revocation”. The details should mention that “the certificate is considered *valid*”.

The presence of revocation information must be checked for the signing certificate and all certificates of its trust chain except for the root certificate.


Certificate Viewer
This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.
Show all certification paths found
QuoVadis Root Certification Auth
QV Schweiz ICA
PDF Tools AG
General
Details
Revocation
Trust
Policies
Legal Notice
The selected certificate is valid
Details
The selected certificate is considered valid because it has not been revoked, as verified using an Online Certificate Status Protocol (OCSP) response that was embedded in the document.
The OCSP Response was signed by QuoVadis OCSP Authority Signature on 2010/03/12 11:00:55 +01'00'
Signer Details...
Problems encountered...
Check revocation
The selected certificate path is valid.
The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'
OK

Time-stamp

The signature can optionally contain a Time-stamp. This is shown in the tab “Date/Time”. The certificate of the Time-stamp server must also be trusted, i.e. its trust chain should be validated as described in the section Trust Chain above.


Signature Properties
Signature is VALID, signed by PDF Tools AG.
Summary
Legal
Signature is timestamped.
Date:
2010/03/12 16:21:08 +01'00'
Timestamps are signed just as documents are signed. For a timestamp signature to be valid you must have trusted the Timestamp Authority that signed the timestamp. Click Show Certificate to view details regarding verification of the timestamp signature.
Timestamp Authority:
SwissSign Platinum TSA - G2 - Unit 0101
Show Certificate...
Timestamps are created with specific policies that are defined by the Timestamp Authority. Amongst other things, a policy can indicate how reliable the time source is. The policy for this timestamp is represented by the identifier 0.0. To understand timestamp policies, you must contact the Timestamp Authority.
Verify Signature
Close

Validation of a PAdES LTV Signature

Verifying if a signature conforms to the PAdES LTV standard is similar to validating a Qualified Electronic Signature.

The following must be checked:

1. Trust Chain
2. Revocation information
3. Time-stamp
4. LTV expiration date
5. Other PAdES Requirements

Trust Chain

Trust chain validation works the same as for validating Qualified Electronic Signatures.

Revocation information

Revocation information (OCPS response or CRL) must be valid and embedded into the signature. In the details, verify that the revocation check was performed using data that was "was *embedded in the signature*". Revocation information that "was contained in the *local cache*" or "was requested *online*" is not embedded into the signature and does not meet PAdES LTV requirements.

Time-stamp

A Time-stamp must be embedded and validated as described for validating Qualified Electronic Signatures. If a document contains multiple Time-stamps, all but the latest one must contain revocation information.

LTV expiration date

The long term validation ability expires with the expiration of the signing certificate of the latest Time-stamp. The life-time of the protection can be further extended beyond the life-of the last Time-stamp applied by adding further DSS information to validate the previous last Time-stamp along with a new Time-stamp.

Other PAdES Requirements

Certain other PAdES requirements, such as requirements on the PKCS#7 CMS, cannot be validated using Adobe Acrobat. For this, use the 3-Heights™ PDF Security API for validation.

6 Reference Manual

This chapter lists all available methods and properties of the 3-Heights™ PDF to PDF/A Converter API. The API provides four interfaces: C, Java, .NET and COM. The following documentation is based on the COM interface. The use of the other interfaces and the names of the functions in these interfaces correspond to the COM interface.

6.1 Pdf2PdfAPI Interface

AddAssociatedFile

```
Method Boolean AddAssociatedFile(String FileName, String Name, Integer Associate,
String AFRelationship, String MimeType, String Description, DATE ModDate)
```

Add a file to the document's embedded files in Convert. For PDF/A-3, the embedded file is associated with an object of the document, i.e. it is an associated file.

This method must be called before Convert. The file is embedded as-is. Embedding files is not allowed for PDF/A-1 and restricted to PDF/A compliant files for PDF/A-2.

Parameters:

- FileName: The path (or URL) to the file to be embedded.
  Name: The name used for the embedded file. This name is presented to the user when viewing the list of embedded files. Default: FileName without the path
- Associate: The object to associate the embedded file with. -1 for none, 0 for document, number greater than 0 for respective page. Default: 0 for PDF/A-3 and -1 otherwise.
- AFRelationship: The relationship of the embedded file to the object associated, PDF/A-3 only. Allowed values are "Source", "Data", "Alternative", "Supplement", and "Unspecified". Default: "Unspecified".
- MimeType: mime-type of the embedded file. Default: "application/octet-stream". Other common values are "application/pdf", "application/xml", or "application/msword".
- Description: A description of the embedded file. This is presented to the user when viewing the list of embedded files.
- ModDate: The modify date of the file. Default: The modify date of the file on the file system or current time, if not available.

Return value:

- True: The file was embedded successfully.
- False: otherwise.

AddEmbeddedFile

```
Method Boolean AddEmbeddedFile(String FileName, String Name)
```

This is a simplified call that is equal to AddAssociatedFile with default arguments. This is for convenience, for example when embedding files in a PDF/A-2 conforming document.

AddZUGFeRDXml

```
Method Boolean AddZUGFeRDXml(String FileName)
Method Boolean AddZUGFeRDXmlMem(Variant MemBlock)
```

Add a ZUGFeRD XML invoice file. Note that this requires the compliance to be set to PDF/A-3.

If the specified ZUGFeRD XML invoice file cannot be added during conversion, a conversion error *ePDFConversionErrorEFRemoved* is generated.

Return value:

- True: The file was found.
- False: Otherwise.

AddFontDirectory

```
Method Boolean AddFontDirectory(String Directory)
```

In order to create a valid PDF/A fonts must be embedded. If the input file contains a font which is not embedded, the font directory is searched for a font with the same name. If such a font is found, the font is embedded.

This method can be used to add (multiple) font directories to the search path for fonts.

Parameters:
- Directory: The path to the font directory that is to be added to the search path.

Return value:
- True: The font directory was added successfully.
- False: otherwise.

If no valid font directory is added, the default font directories are used. The location of these directories depends on the operating system:

- Windows: %SystemRoot%\Fonts and directory "Fonts", which must be a direct sub-directory of where the main DLL or executable resides.
- Mac: /System/Library/Fonts and /Library/Fonts
- Unix: $PDFFONTDIR or /usr/lib/X11/fonts/Type1

See chapter Fonts for more information on the font directories and font handling of the 3-Heights™ PDF to PDF/A Converter API in general.

AnalyzeOnly

```
Property Boolean AnalyzeOnly
Accessors: Get, Set
Default: False
```

When set to true, the method Convert() analyzes the input file only, it does not create an output file. The results of the analysis are written to the corresponding log file. If Convert() returns True and the ErrorCode is not PDF_E_CONFORMANCE, the file is compliant.

The analysis is equal to the analysis using the 3-Heights™ PDF Validator and validating against PDF/A.

BeginSession

```
Method Boolean BeginSession(String Provider)
```

The methods BeginSession and `EndSession` support bulk digital signing by keeping the session to the security device (HSM, Token or Cryptographic Provider) open. See the chapter Guidelines for Mass Signing for more guidelines.

For backwards compatibility the use of these methods is optional. If used, the Provider property may not be set. If omitted an individual session to the provider indicated by the property Provider is used for each signature operation.

Parameters:
- Provider: See property `Provider`.

Return value:
- True: Session started successfully
- False: Otherwise

CertificateName

```
Property String CertificateName
Accessors: Get, Set
Default: ""
```

Set or get the name of the certificate.

The "CertificateName" corresponds to the common name (CN) of the subject. In the Windows' certificate store this corresponds to "Issued to".

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

ColorSpaceProfile

```
Property String ColorSpaceProfile
Accessors: Get (after Convert), Set (before Convert)
Default: (Windows only)
%SystemRoot%\system32\spool\drivers\color\USWebCoatedSWOP.icc
```

This property us used to set and get ICC color profile file names. ICC profiles can be set prior to Convert and read after Convert.

Setting ICC profiles makes the converter substitute device color spaces with ICC based color spaces. At most three ICC profiles can be set, as substitutes for DeviceRGB, DeviceCMYK, and DeviceGray respectively. To set several ICC profiles, set this property several times. The matching device color space to be substituted is selected automatically. Note that it is not necessary to set ICC profiles for a successful conversion, see Section 5.3.

When getting this property, only one ICC profile file name can be queried: If the output intent is set to an RGB profile (a CMYK profile) then the file name of any set CMYK profile (RGB profile respectively) results. Otherwise any set CMYK or RGB profile results with CMYK taking precedence.

If a required color space profile is not available, a conversion error is generated.

C and Java interfaces only: If the provided path is not a valid profile, the method fails.

Compliance

```
Property TPDFCompliance Compliance
Accessors: Get, Set
Default: ePDFA2b
```

This property sets or gets the compliance level of the output PDF. Supported values for the enumeration `TPDFCompliance` are:

- `ePDFA1a`
- `ePDFA1b`
- `ePDFA2a`
- `ePDFA2b`
- `ePDFA2u`
- `ePDFA3a`
- `ePDFA3b`
- `ePDFA3u`

Other listed entries (e.g. ePDF10, ePDF11...ePDF17, ePDFUnk) are not supported as output compliance level by the 3-Heights™ PDF to PDF/A Converter API.

Some files cannot be converted to the compliance requested. The 3-Heights™ PDF to PDF/A Converter API can detect this and up- or downgrade the compliance automatically. Use the property `ConversionErrorMask` to prevent automatic up- or downgrades.

ConversionErrors

```
Property Long ConversionErrors
Accessors: Get
```

Get conversion error events that occurred during conversion. This property should be queried after Convert or ConvertMem returned true and the property ErrorCode is set to PDF_E_CONVERSION.

See enumeration TPDFConversionError for a list of supported conversion error events.

ConversionErrorMask

```
Property TPDFConversionError ConversionErrorMask
Accessors: Get, Set
Default: ePDFConversionErrorVisualDiff + ePDFConversionErrorOCGRemoved +
         ePDFConversionErrorTranspRemoved + ePDFConversionErrorEFRemoved +
         ePDFConversionErrorXMPRemoved + ePDFConversionErrorLinearization +
         ePDFConversionErrorSigCrea + ePDFConversionErrorCorrupt
```

Define which conversion operations shall result in a conversion error, i.e. set the return value of Convert or ConvertMem to true and the property ErrorCode after the conversion to PDF_E_CONVERSION. In the case of a conversion error, use the property ConversionErrors to retrieve the actual errors of the conversion.

See enumeration TPDFConversionError for a list of supported conversion error events.

Convert

```
Method Boolean Convert(String InputFileName, String Password,
String OuputFileName, String LogFileName)
```

Open a PDF document from file, convert it to PDF/A and save the output document to a file.

Parameters:

- InputFileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules of the input file.
- Password (optional): The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.
- OutputFileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules of the output file.
- LogFileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules of the log file.

Return value:

- True: If the function executed successfully, i.e. could read the input file and write an output file.
  The correctness of the resulting output file is not reflected by the return value, but the property ErrorCode which may have the one of following values:
  - PDF_E_POSTANALYSIS: The post analysis detected an error. This means, that the output file does not conform to the requested standard. Note that post analysis is active only, if the property PostAnalyze is set to true. This error has priority over PDF_E_CONVERSION, so if PostAnalyze is set and ErrorCode is not PDF_E_POSTANALYSIS, the output file conforms to the requested standard.
  - PDF_E_CONVERSION: A conversion error occurred. The conditions that can lead to this error can be defined using the property ConversionErrorMask. The error conditions occurred are indicated by the property ConversionErrors. The conformance of the resulting output document is unrelated to this error.

  An overview of the conversion process is provided in section “Process Description”.
- False: If no valid output document is written. Check the error code to get the cause of the problem.
  Possible causes are:
  - Lincense is not set or invalid
  - Input file does not exist
  - Input file is protected by a user password and the provided password is incorrect
  - Output file is not writable, e.g. locked
  - Conversion was stopped
  - The input file is not a PDF file or contains unrendered XFA fields.

ConvertAlways

```
Property Boolean ConvertAlways
Accessors: Get, Set
Default: False
```

Prior to the conversion, an analysis of the input file is performed. The analysis checks for conformance violations which inhibit a conversion.

A conversion to PDF/A can fail if the document stands in conflict with one of the following issues:

- Non-conformance with the PDF Reference
- Layers (PDF/A-1 only)
- Transparency (PDF/A-1 only)
- Missing or ambiguous annotation and form field appearances
- Unknown annotation types (optional)
- Multimedia annotations (optional)

Under the following circumstances the conformance level is automatically downgraded:

- Missing Unicode information: Downgrade to level B, e.g. from pdfa-2a to pdfa-2b
- Missing logical structure: Downgrade to level B (for PDF/A-1) or level U (for PDF/A-2 and PDF/A-3). E.g. from pdfa-1a to pdfa-1b

Setting the property *ConvertAlways* to true forces the conversion even if there is a problem with the input file or the input file is already compliant. The created output may or may not be compliant with the requested PDF version.

ConvertMem

```
Method Boolean ConvertMem(Variant vin, String Password, Variant vout, Variant vlog)
```

Open a PDF document from memory, convert it to PDF/A and write the output document to a byte array.

Parameters:

- vin: A byte array containing the input PDF document.
- Password (optional): The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.
- vout: A byte array to which the output PDF/A document is written.
- vlog: A byte array to which the log is written.

Return value:

- Same as function Convert.

EmbedT1asCFF

```
Property Boolean EmbedT1asCFF
Accessors: Get, Set
Default: False
```

Convert Type1 (PostScript) fonts to Compact Font Format before embedding. This reduces the file size. This affects the embedding of fonts only, existing Type1 fonts of the input document will not be converted.

EmbedRevocationInfo

```
Property Boolean EmbedRevocationInfo
Accessors: Get, Set
Default: True
```

Revocation information is either an OCSP response or a CRL, which is provided by a validation service at the time of signing and acts as proof that at the time of signing the certificate is valid. This is useful because even when the certificates expires or is revoked at a later time, the signature in the signed document remains valid.

Embedding revocation information is optional but suggested when applying advanced or qualified electronic signatures.

The downsides of embedding revocation information are the increase of the file size (normally by around 20k) and that it requires a connection to a validation service, which delays the process of signing (normally by around 2 seconds). For mass signing it is suggested to use the caching mechanism, see chapter "Caching of CRLs, OSCP and TSP Responses".

EndSession

```
Method Boolean EndSession()
```

Ends the opten session to the security device.

See BeginSession.

ErrorCode

```
Property TPDFErrorCode ErrorCode
Accessors: Get
```

This property can be accessed to receive the latest error code. See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file *pdferror.h*. Please note, that only few of them are relevant for the 3-Heights™ PDF to PDF/A Converter API.

ExportText

```
Method Boolean ExportText(String FileName)
```

Export the retrieved OCR text to a file. This function can only be used in combination with an OCR engine (see property SetOCREngine). When an OCR engine is set, the OCR text is always embedded in the resulting PDF document. If this method is used, it is in addition also extracted to a file.

The output format is a table, where rows are separated by a new line and columns are separated by a tabulator.

The table contains the following columns:

`Page`	Page number
`Image`	PDF object number which contains the image
`FontSize`	Font size in points
`FontName`	Font name, for any barcode font the name is "Barcode". This value is only set if the font name is returned by the OCR engine.
`FontFamily`	1: Serif 2: SansSerif 3: Monospaced This value is only set if provided by the OCR engine.

`FontStyles`	1: Barcode 2: Bold 4: Italic 8: Underline 16: Strikeout This value is only set if provided by the OCR engine. Example: 6 = 2 + 4 = Bold + Italic
`Baseline`	Baseline of the text
`Left, Top, Right, Bottom`	Bounding box of the text in PDF coordinates
`String`	Recognized text

FlattenSignatures

```
Property Boolean FlattenSignatures
Accessors: Get, Set
Default: False
```

Remove all signed signature fields and add their appearances to the page's content. Note that the signatures themselves (the cryptographic parts) are removed and hence the bit ePDFConversionErrorDocSigned of the conversion error is set regardless of the value of the FlattenSignatures property.

Processing the PDF with 3-Heights™ PDF to PDF/A Converter API breaks existing signatures and their cryptographic parts need to be removed. In general, the visual appearances of signatures are regarded as worthless without the cryptographic part and are therefore removed by default as well. The visual appearances can be preserved by setting this property FlattenSignatures to True.

GetOCRPluginCount

```
Method Integer GetOCRPluginCount()
```

OCR engines are accessed through the corresponding OCR interface DLLs. At present there are three OCR engine supported:

Abbyy FineReader 8.1 OCR Engine: This engine is accessed by the OCR interface DLL *pdfocrpluginAbbyy.ocr*.

3-Heights™ OCR Service: This service is accessed by the OCR interface DLL *pdfocrpluginService.ocr*. The service accesses the Abbyy FineReader 8.1 OCR Engine.

Tesseract: This engine is accessed by the OCR interface DLL *pdfocrpluginTesseract.ocr*.

The OCR interface DLL is provided by the 3-Heights™ PDF to PDF/A Converter API.

The OCR engine is provided as a separate product: 3-Heights™ OCR Enterprise Add-On.

In order to make use of the OCR engine, the OCR interface DLL and the OCR engine must be installed. The property *GetOCRPluginCount* returns the number of available OCR interface DLLs. It does not verify the corresponding OCR engines are installed and can be initialized. The OCR engine is loaded with the method *SetOCREngine*.

Return value:
- The number of available OCR engines (i.e. their corresponding OCR interface DLLs).

GetOCRPluginName

```
Method String GetOCRPluginName(Integer iOCREngine)
```

An OCR engine is accessed through an OCR plug-in. Each plug-in corresponds to one OCR engine. The number of OCR plug-ins is retrieved using n = GetOCRPluginCount. The method GetOCRPluginName(n) returns the name of the nth OCR Engine which corresponds to that OCR plug-in. At present there are three OCR engines available: “Abbyy”, “service” and “tesseract”.

Parameters:
- iOCREngine: The nth OCR engine. The total number of engines is retrieved using GetOCRPluginCount.

Return value:
- The name of the nth OCR engine. Null if it doesn’t exist.

ImageQuality

```
Property Integer ImageQuality
Accessors: Get, Set
Default: 75
```

Set or get the image quality index for images that use a prohibited lossy compression type are must be recompressed.

Example: JPX is not allowed in PDF/A-1. If a PDF contains a JPX compressed image, its compression type must be altered. Thus the 3-Heights™ PDF to PDF/A Converter API converts it to an image with regular JPEG compression and the image quality as defined by this property.

Supported values are 1 to 100. A higher value means better visual quality at the cost of a larger file size. Recommended values range from 70 to 90. The default value is 75.

InfoEntry

```
Property String InfoEntry(String Key)
Accessors: Get, Set
```

Parameters:
- Key: A key as string

Return value:
- Value: The value as string

Retrieve or add a key-value pair to the documents info object. Values of predefined keys are also stored in the XMP metadata package.

Popular entries defined by the PDF Specification and used by most PDF viewers are “Title”, “Author”, “Subject”, “Creator” (sometimes referred to as Application) and “Producer” (sometimes referred to as PDF Creator).

Examples in Visual Basic 6:

- Get document title: `t = InfoEntry("Title")`
- Set document title: `InfoEntry( "Title ") = "my title"`
- Set the creation date to 13:55:33, April 5, 2010, UTC+2:
  `InfoEntry("CreationDate") = "D: 20100405135533 + 02'00'"`

Issuer

```
Property String Issuer
Accessors: Get, Set
Default: ""
```

Set the issuer of the certificate. The “Issuer” corresponds to the common name (CN) of the issuer. In the Windows’ certificate store this corresponds to “Issued by”.

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

Linearize

```
Property Boolean Linearize
Accessors: Get, Set
Default: False
```

Get or set whether to linearize the PDF output file for fast web access.

Linearization is the process of preparing a PDF file in a way that permits random page access by a web browser. While the whole non-linearized PDF file must be downloaded before the first page can be displayed, this is not the case for a linearized file.

NoCache

```
Property Boolean NoCache
Accessors: Get, Set
Default: False
```

Get or set whether to disable the cache for CRL and OCSP responses.

Using the cache is safe, since the responses are cached as long as they are valid only. The option affects both signature creation and validation.

See section on caching for more information on the caches.

OCRBitonalRecognition

```
Property Boolean OCRBitonalRecognition
Accessors: Get, Set
Default: false
```

Specifiy whether the images should be converted to bi-tonal (black and white) before OCR recognition.

Enabling this feature can improve the memory consumption of the OCR process. It is suggest to set this property to true when using ABBYY 8 or Tesseract.

Enabling this feature automatically re-embeds the original images in the output document. The setting of the property OCRReembedImages is therefore ignored.

OCRMode

```
Property Integer OCRMode
Accessors: Get, Set
Default: 1
```

Specify behavior of converter for files with existing OCR text. Available OCR modes are the following:

1 Only perform OCR for images without existing OCR text (default).

2 If OCR engine is active, remove old OCR text and perform OCR for all images. Hence, existing OCR text is not removed, if OCR engine is not active.

3 Always remove old OCR text and, if OCR engine is active, perform OCR for all images. This can be used to strip existing, without adding new OCR text.

4 Only perform OCR, if input file contains no text.

OCRReembedImages

```
Property Boolean OCRReembedImages
Accessors: Get, Set
Default: True
```

The OCR engine de-skews and de-noises the input image before recognizing the characters. This option controls, whether the 3-Heights™ PDF to PDF/A Converter API should use the de-skewed image or keep the original image.

`True` Embed the de-skewed and de-noised image from the OCR engine. This might change the appearance of the page. This setting is recommended for scanned documents.

`False` Embed the original image (also see property OCRDeskewImage). This setting is recommended for born-digital documents.

OCRDeskewImage

```
Property Boolean OCRDeskewImage
Accessors: Get, Set
Default: True
```

Do not de-skew original image (if OCRReembedImages is set to True only). This option specifies whether the image and text are de-skewed according to the recognized skew angle.

`True` Rotate image, such that lines of text are made horizontal. This might change the appearance of the page. This setting is recommended for scanned documents.

`False` Do not change skew of images. This setting is recommended for born-digital documents.

OCREmbedBarcodes

```
Property Boolean OCREmbedBarcodes
Accessors: Get, Set
Default: False
```

This property specifies whether the recognized barcodes are embedded in the XMP metadata.

OCRRotatePage

```
Property Boolean OCRRotatePage
Accessors: Get, Set
Default: False
```

This property specifies whether the page is rotated according to the recognized image rotation.

OCRResolutionDPI

```
Property Single OCRResolutionDPI
Accessors: Get, Set
Default: 300
```

Resample images to target resolution before they are sent to the OCR engine. The default is 300 dpi, which is the preferred resolution for most OCR engines.

OCRThresholdDPI

```
Property Single OCRThresholdDPI
Accessors: Get, Set
Default: 400
```

Only images with a higher resolution than the threshold are re-sampled before OCR. The default is 400 dpi. If set to -1: no re-sampling is applied.

OutputIntentProfile

```
Property String OutputIntentProfile
Accessors: Get, Set
Default: (Windows only)
    %SystemRoot%\system32\spool\drivers\color\USWebCoatedSWOP.icc
```

Set or get the path to the ICC profile for the output intent. This property can be set prior to Convert and can be read after Convert. See Section 5.3 for the usage of the output intent.

If an invalid path is provided, Convert fails and writes a corresponding message to the log file.

The given profile is embedded only if the input file does not contain a PDF/A output intent already.

If during conversion an output intent was set automatically, then the path for this profile can be queried after conversion by getting this property.

C and Java interface only: If the provided path is not a valid profile, the method fails.

PostAnalyze

```
Property Boolean PostAnalyze
Accessors: Get, Set
Default: False
```

Analyze the created PDF output file and verify if it meets the specified compliance level. The result of this analysis is written to the log file. If Convert() returns True and the ErrorCode is not PDF_E_POSTANALYSIS, the output file is compliant.

The post analysis is executed only, if an output file was created and the conversion was successful. The property PostAnalysis is ignored if the property AnalyzeOnly is true.

The post analysis can detect errors in the created output file that could not be predicted based on the analysis of the input file nor could they be detected during the conversion, because the conversion also depends on the input parameters (such as ICC profiles).

The post-analysis is equal to the analysis using the 3-Heights™ PDF Validator and validating against PDF/A.

Provider

```
Property String Provider
Accessors: Get, Set
Default (Windows only): "Microsoft Base Cryptographic Provider v1.0"
```

This property specifies the cryptographic provider used to create and verify signatures.

For more information on the different providers available, see the description in the respective subsection of the section Cryptographic Provider.

- When using the Microsoft CryptoAPI Provider, the value of this property with the following syntax:

  ```
  "[ProviderType:]Provider[;PIN]"
  ```

  Examples, “123456” being the pin code:
  - `Provider = "Microsoft Base Cryptographic Provider v1.0;123456"`
  - `Provider = ";123456"`

- When using the PKCS#11 Provider, the value of this property is to be set to a string with the following syntax:

  ```
  "PathToDll;SlotId;Pin"
  ```

  Examples:
  - `Provider = "\WINDOWS\system32\siecap11.dll;4;123456"`

- When using any of the service providers, such as the Swisscom All-in signing service, the value of this property is essentially the url of the service endpoint:

  ```
  "http[s]://server.servicedomain.com:8080/url"
  ```

ProxyURL

```
Property String ProxyURL
Accessors: Get, Set
Default: ""
```

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

ProxyCredentials

```
Property String ProxyCredentials
Accessors: Get, Set
Default: ""
```

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

Remove Signature

```
Property Boolean RemoveSignature
Accessors: Get, Set
Default: True
```

(deprecated) If set to true and the input document is not PDF/A compliant, all signatures (including MDP signature) and signature appearances of the input document are removed.

If set to false and the input document is not PDF/A compliant, the conversion is aborted; Convert() returns false and Errorcode is set to PDF_E_DOCSIGNED.

If the input document is PDF/A compliant, the document is not converted and the existing signatures remain. Optionally an additional signature can be applied.

This property is deprecated, instead use:

TPDFConversionError.EPDFConversionErrorDocSigned.

ReportDetails

```
Property Boolean ReportDetails
Accessors: Get, Set
Default: False
```

Write a detailed list of errors and warnings from the analysis of the input file as well as the optional post-analysis of the output file to the log file.

Setting this property to true, the conversion step lists all conformance violations per page. Each violation is listed with a page number (page 0 = document level), error number, a description and a counter of how many times the error occurs. The option provides more detailed information than ReportSummary. All errors are listed in the header file *pdferror.h*. Here are a few examples of possible errors:

- 0, 0x80410604, "The key Metadata is required but missing.", 1
- 0, 0x80410604, "The key MarkInfo is required but missing.", 1
- 1, 0x00418704, "The font Arial-BoldMT must be embedded.", 1
- 1, 0x83410612, "The document does not conform to the requested standard.", 1

ReportSummary

```
Property Boolean ReportSummary
Accessors: Get, Set
Default: False
```

Write a summary of errors and warnings from the analysis of the input file as well as the optional post-analysis of the output file to the log file. If any of the following violations is detected at least once, it is reported (once). This report provides less detailed information than the detailed list per page provided by *ReportDetails*.

- The file format (header, trailer, objects, xref, streams) is corrupted.
- The document doesn't conform to the PDF reference (missing required entries, wrong value types, etc.).
- The file is encrypted and the password was not provided.
- The document contains device-specific color spaces.
- The document contains illegal rendering hints (unknown intents, interpolation, transfer and halftone functions).
- The document contains alternate information (images).
- The document contains embedded PostScript code.
- The document contains references to external content (reference XObjects, file attachments, OPI).
- The document contains fonts without embedded font programs or encoding information (CMAPs).
- The document contains fonts without appropriate character to Unicode mapping information (ToUnicode maps).
- The document contains transparency.
- The document contains unknown annotation types.
- The document contains multimedia annotations (sound, movies).
- The document contains hidden, invisible, non-viewable or non-printable annotations.
- The document contains annotations or form fields with ambiguous or without appropriate appearances.
- The document contains actions types other than for navigation (launch, JavaScript, ResetForm, etc.).

- The document's meta data is either missing or inconsistent or corrupt.
- The document doesn't provide appropriate logical structure information.
- The document contains optional content (layers).

SetMetadata

```
Method Boolean SetMetadata(Single FileName)
```

Set the document's XMP metadata. The XMP metadata is inserted as is, which means it is not parsed and validated. If no XMP metadata is provided, the 3-Heights™ PDF to PDF/A Converter API generates it automatically.

Parameters:
- FileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules of the file containing the XMP metadata.

Return value:
- True: The XMP metadata file was set successfully.
- False: otherwise.

SetOCREngine

```
Method Boolean SetOCREngine(String Engine)
```

This method require the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

Set the OCR engine that is used when OCR information shall be added during the conversion. If the engine's name is set to an empty string, OCR is not applied.

See also documentation for the 3-Heights™ OCR Add-On.

Parameters:
- Engine: The name of the OCR engine (e.g. "abbyy"). For every available OCR engine, there is a corresponding OCR interface DLL. The OCR interface DLLs (e.g. *pdfocrAbbyy.ocr*) are distributed with the 3-Heights™ PDF to PDF/A Converter API and are required to communicate with the OCR engine. The names of all available OCR engines can be retrieved using the properties *GetOCREngineCount* and *GetOCREngine*.

Return value:
- True: The OCR interface DLL was found, the OCR engine was found and the OCR engine was successfully initialized.
- False: otherwise.

SetOCRLanguages

```
Method Boolean SetOCRLanguages(String Languages)
```

This method requires the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

Setting a languages helps the OCR engine to minimize errors by means of using dictionaries of the defined languages.

This method must be called after SetOCREngine.

If SetOCRParams is used, *SetOCRLanguages* must be called after SetOCRParams.

Parameters:
- Languages: A string of one or multiple, comma-separated languages. The supported names are OCR engine dependant. The OCR engine will only use dictionaries of the set languages.

Return value:
- True: The Language(s) were successfully set
- False: Otherwise

Example

```
SetOCREngine("abbyy")
SetOCRLanguages("English, German")
```

SetOCRParams

```
Method Boolean SetOCRParams(String Params)
```

This method requires the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

By means of this method, OCR engine specific settings can be applied in the form of key-value pairs. These pairs are OCR engine dependant and are described in the corresponding manual.

Parameters:
- Params: A list of comma-separated key value pairs. See example.

Return value:
- True: The OCR parameters were successfully set.
- False: otherwise.

Example

```
SetOCREngine("abbyy")
SetOCRParams("BalancedMode=TRUE, DetectBold=FALSE")
```

SetSessionProperty

```
Method Boolean SetSessionPropertyString(String Name, String Value)
Method Boolean SetSessionPropertyBytes(String Name, Variant Value)
```

Provider-specific session configuration.

Properties have to be set before calling BeginSession and are deleted when calling EndSession.

Parameters:
- Name: The name of the property. The names that are supported are specific to the provider used with BeginSession.
- Value: The value of the property as string or byte array.

SetSignatureRectangle

```
Method Void SetSignatureRectangle(Single x, Single y, Single w, Single h)
```

Set the position and size of the digital signature annotation. The default is in the lower left corner. The units are PDF points (1 point = 1/72 inch, A4 = 595x842 points, Letter = 612x792 points) measured from the lower left corner of the page.

The position is defined by the four values for the lower-left corner (x, y) and dimensions (w, h) of the rectangle. If either the width or height is zero or negative, an invisible signature is created, i.e. no visible appearance is created for the signature. To create an invisible signature call `SetSignatureRectangle(0, 0, 0, 0)`.

SetToUnicodeFile

```
Method Void SetToUnicodeFile(String FileName)
```

Update the fonts' Unicodes as specified by file. The file must contain the mapping of character codes to Unicodes for specific fonts.

SignatureImageFileName

```
Property String SignatureImageFileName
Accessors: Get, Set
Default: ""
```

Path to background image of signature appearance. The image is centered and scaled down proportionally to fit into the given rectangle. If the path is NULL, or the image does not exist, the appearance's background is a filled rectangle using the colors FillColor and StrokeColor. Note that for the output file to be PDF/A, the image's color space must match the document's output intent.

SignatureLocation

```
Property String SignatureLocation
Accessors: Get, Set
Default: ""
```

This is the physical location where the signature was added, for example "Zurich, Switzerland".

If this property is optional and not added if set to an empty string.

SignaturePageNo

```
Property Long SignaturePageNo
Accessors: Get, Set
Default: -1 (last page)
```

The page number where the signature is to be added to the document. If an invalid page number is set, it is added to the last page.

SignatureReason

```
Property String SignatureReason
Accessors: Get, Set
Default: ""
```

Add a descriptive text for the digital signature. A text is not required in order to create a valid signature.

SignatureSerialNumber

```
Property String SignatureSerialNumber
Accessors: Get, Set
Default: ""
```

The serial number with the issuer can be used to select a certificate for signing. This property is a hex string as displayed by the "Serial number" field in the Microsoft Management Console (MMC), e.g. "49 cf 7d d1 6c a9".

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

SignerContactInfo

```
Property String SignerContactInfo
Accessors: Get, Set
Default: ""
```

Add a descriptive text as signer contact info, e.g. a phone number. This enables a recipient to contact the signer to verify the signature. This is not required in order to create a valid signature.

If this property is set to an empty string, no entry is created.

SignerFingerprint

```
Property Variant SignerFingerprint
Accessors: Get, Set
```

The sha1 fingerprint of the signer certificate. This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use). After validating a signature, this property contains the validated signature's fingerprint.

SignerFingerprintStr

```
Property String SignerFingerprintStr
Accessors: Get, Set
```

The hex string representation of the signer certificate's sha1 fingerprint. This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

All characters outside the ranges 0-9, a-f and A-F are ignored. In the Microsoft Management Console, the "Thumbprint" value can be used without conversion, if the "Thumbprint algorithm" is "sha1". E.g. "b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5".

Store

```
Property String Store
Accessors: Get, Set
Default: "MY"
```

For the Microsoft CryptoAPI Provider this defines the certificate store from where the signing certificate should be taken. This depends on the OS. The default is "MY". Other supported values are: "CA" or "ROOT".

StoreLocation

```
Property Integer StoreLocation
Accessors: Get, Set
Default: 1
```

For the Microsoft CryptoAPI Provider this defines the location of the certificate store from where the signing certificate should be taken. Supported are:

0 Local Machine

1 Current User (default)

For more information, see the detailed description of the Microsoft CryptoAPI Provider.

SubsetFonts

```
Property Boolean SubsetFonts
Accessors: Get, Set
Default: TRUE
```

By default, fonts that are embedded are automatically subset to minimize the file size. If for any reason, e.g. post-processing, fonts shall not be subset, set the property *SubsetFonts* to false . Whether fonts are subset or not is irrelevant with respect to the compliance with PDF/A. (Relevant is only that all used glyphs are contained in the font program).

TimeStampCredentials

```
Property String TimeStampCredentials
Accessors: Get, Set
Default: ""
```

If a Time-stamp server requires authentication, use this property to provide the credentials. Credentials commonly have the syntax “username:password”.

TimeStampURL

```
Property String TimeStampURL
Accessors: Get, Set
Default: ""
```

The URL of the trusted Time-stamp authority (TSA) from which a Time-stamp shall be acquired. This setting is suggested to be used when applying a Qualified Electronic Signature. Example: “tsu.my-timeserver.org”. Applying a Time-stamp requires an online connection to a time server; the firewall must be configured accordingly. In case a web proxy is used, it must be ensured the following MIME types are supported:

`application/timestamp-query`

`application/timestamp-reply`

If an invalid Time-stamp server address is provided or no connection can be made to the time server, the return code of SaveAs is true, and the property ErrorCode is set to SIG_CREA_E_TSP after calling SaveAs.

Terminate

```
Method Void Terminate
```

Terminate all open sessions, and finalize and unload all PKCS#11 drivers. Some drivers require Terminate to be called. Otherwise, your application might crash and/or your HSM, USB token or smart card might not be unlocked.

When using the C/C++ API, Terminate may not be called from the context of the destructor of a global or static object, an atexit() handler, nor the DllMain() entry point.

Make sure to end all open sessions and dispose of all Pdf2Pdf objects before calling Terminate. After calling Terminate, the process may not call any other methods of this class.

TestSession

```
Method Boolean TestSession()
```

Test if the current session is still alive.

Return value:

- True: Subsequent calls to Convert and ConvertMem are likely to succeed.
- False: Subsequent calls to Convert and ConvertMem are unlikely to succeed. Error codes are the same as in Convert where applicable.

6.2 Enumerations

Note: Depending on the interface, enumerations may have “TPDF” as prefix (COM, C) or “PDF” as prefix (.NET) or no prefix at all (Java).

TPDFCompliance

ePDF10	PDF Version 1.0
ePDF11	PDF Version 1.1
ePDF12	PDF Version 1.2
ePDF13	PDF Version 1.3
ePDF14	PDF Version 1.4 (corresponds to Acrobat 5)
ePDF15	PDF Version 1.5
ePDF16	PDF Version 1.6 (corresponds to Acrobat 7)
ePDF17	PDF Version 1.7
ePDFA1a	PDF/A 1a, ISO 19005-1, Level A compliance
ePDFA1b	PDF/A 1b, ISO 19005-1, Level B compliance
ePDFA2a	PDF/A 2a, ISO 19005-2, Level A compliance
ePDFA2b	PDF/A 2b, ISO 19005-2, Level B compliance
ePDFA2u	PDF/A 2u, ISO 19005-2, Level U compliance
ePDFA3a	PDF/A 3a, ISO 19005-3, Level A compliance
ePDFA3b	PDF/A 3b, ISO 19005-3, Level B compliance
ePDFA3u	PDF/A 3u, ISO 19005-3, Level U compliance
ePDFUnk	Unknown format (default)

TPDFErrorCode

All TPDFErrorCode enumerations start with "PDF_" followed by a single letter which is one of "S", "E", "W" or "I", an underscore and a descriptive text. The single letter gives in an indication of the type of error. These are: Success, Error, Warning, Information. With respect to corrupt PDF files: An error indicates a corruption in the PDF, the file may or may not be readable. A warning indicates the file is readable but not valid.

A full list of all PDF Tools error codes is available in the header file *pdferror.h*. Note that only a few are relevant for the PDF to PDF/A Converter API. The most common are listed here.

`PDF_S_SUCCESS`	The operation was completed successfully.
`LIC_E_NOTSET`, `LIC_E_NOTFOUND`, …	Various license management related errors.
`PDF_E_FILEOPEN`	Failed to open the file.
`PDF_E_FILECREATE`	Failed to create the file.
`PDF_E_PASSWORD`	The authentication failed due to a wrong password.
`PDF_E_UNKSECHANDLER`	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.
`PDF_E_XFANEEDSRENDERING`	The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file.
`PDF_E_CONFORMANCE`	The document does not conform to the requested standard (pre analysis).
`PDF_E_CONVERSION`	Output file has conversion errors. For more information see section "Process Description" and the description of the method Convert.
`PDF_E_POSTANALYSIS`	Output file is not conformant (post analysis). For more information see section "Process Description" and the description of the method Convert.
`PDF_E_STOPPED`	Cannot convert input file due to conformance problems. The input file is probably corrupt.

TPDFConversionError

ePDFConversionErrorNone	None
ePDFConversionErrorUpgrade	Set this flag to prevent automatic upgrades from PDF/A-1 to PDF/A-2. When converting a file to PDF/A-1, transparency needs to be removed. This may cause significant visual differences. Therefore, the default behavior of the 3-Heights™ PDF to PDF/A Converter API is to convert the file to PDF/A-2.
ePDFConversionErrorDowngrade	Set this flag to prevent automatic downgrades of the conformance level, e.g. from PDF/A-1a to PDF/A-1b. If this flag is not set, the level is downgraded under the following conditions: ▪ Downgrade to level B: If a file contains text that is not extractable (i.e. missing ToUnicode information). Example: Downgrade PDF/A-2u to PDF/A-2b. ▪ Downgrade to level U (PDF/A-2 and PDF/A-3) or B (PDF/A-1): Level A requires logical structure information ("tagging") information, so if a file contains no such information, its level is downgraded. Logical structure information in a PDF defines the structure of content, such as titles, paragraphs, figures, reading order, tables or articles. Logical structure elements can be "tagged" with descriptions or alternative text. "Tagging" allows the contents of an image to be described to the visually impaired. It is not possible for the 3-Heights™ PDF to PDF/A Converter API to add meaningful tagging information. Adding tagging information without prior knowledge about the input file's structure and content is neither possible nor allowed by the PDF/A standard. For that reason, the conformance level is automatically downgraded to level B or U. Example: Downgrade PDF/A-1a to PDF/A-1b.
ePDFConversionErrorVisual-Diff	Visual differences in output file.
ePDFConversionErrorColorants	Resolve name collisions of colorants (PDF/A-2 and PDF/A-3 only).
ePDFConversionErrorOCG-Removed	Remove optional content groups (layers) (PDF/A-1 only).
ePDFConversionErrorTransp-Removed	Remove transparency (PDF/A-1 only).
ePDFConversionErrorEFRemoved	Remove embedded files.
ePDFConversionErrorXMP-Removed	Remove non convertible XMP metadata.
ePDFConversionError-Linearization	Error during linearization of output file.
ePDFConversionErrorDocSigned	Conversion of signed document forced removal of signatures. The conversion of a file to PDF/A invalidates all signatures of the input file. For that reason, all signatures are removed and this conversion event is set. Optionally the visual appearances of the signatures can be preserved by setting the property FlattenSignatures to true.
ePDFConversionErrorSigCrea	Failed to create signature.
ePDFConversionErrorNetwork	Failed to add revocation information (OCSP or CRL) or TSP to signature.

`ePDFConversionErrorCorrupt`	The input document is corrupt and should be repaired. The errors encountered are printed to the log file. Some errors can be repaired, but it is crucial to review the output file and perform the post analysis.
`ePDFConversionErrorOCR`	OCR error occurred.
`ePDFConversionErrorFontSubst`	Failed to find the same font for embedding, so a similar substitution font was used.

7 Log File

All steps in the diagram from chapter "Process Description" can write to the log file. There are three types of messages in the log file: Warnings/Information, Errors and Reports.

Warnings and Information

describe the current process step. They do not inhibit the conversion. Prefix: -

Example:

```
- Opening file input.pdf
- Setting font directory to C:\WINNT\Fonts
- Analyzing input.pdf
- Conformance level A has been downgraded to level B
```

Errors

inhibit a successful conversion. Prefix: *

Example:

```
* Cannot open file input.pdf.
Input file 001.pdf isn't convertible.
```

Reports

are only created if the corresponding option (Details or Summary) is selected. Prefix: none

Example: Details

```
0, 0x80410604, "The key Metadata is required but missing.", 1
2, 0x00418704, "The font Verdana must be embedded.", 1
2, 0x83410614, "The document contains device-specific color spaces (Annotation C or IC).", 1
```

Example: Summary

```
The document contains fonts without embedded font programs or encoding information (CMAPs).
The document's meta data is either missing or inconsistent or corrupt.
The document doesn't provide appropriate logical structure information.
```

8 Licensing and Copyright

The 3-Heights™ PDF to PDF/A Converter API is copyrighted. This user's manual is also copyright protected; it may be copied and given away provided that it remains unchanged including the copyright notice.

9 Contact

PDF Tools AG

Kasernenstrasse 1

8184 Bachenbülach

Switzerland

http://www.pdf-tools.com